Программный комплекс Систэм Платформ

Язык SePlatform.Om 1.1

Общее описание

Редакция	Соответствует версии ПО
3. Предварительная	1.1.7

Содержание

1. Введение

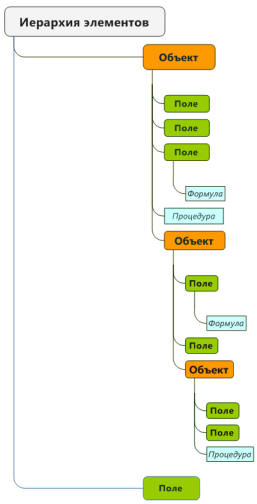
В документе описан синтаксис и возможности языка SePlatform.Om. Сведения, приведенные в данном документе, являются общими для всех продуктов, в которых применяется язык.

1.1. Окружение и модель данных

Язык SePlatform.Om позволяет описывать исполняемые элементы скриптов (процедуры и формулы), которые запускаются в процессе работы компонентов Систэм Платформ. Скрипты можно создавать в процессе работы со следующими программными продуктами:

- SePlatform.Data Server
- SePlatform.HMI
- SePlatform.Development Studio

Исполняемые элементы (процедуры и формулы) в процессе выполнения обращаются к данным. Представление элементов данных зависит от компонента Систэм Платформ, в окружении которого работает скрипт. Однако базовая структура иерархии элементов данных и исполняемых элементов является одинаковой для любого окружения - будь то окружение сигнальной модели данных в SePlatform.Data Server или объектной модели данных сред SePlatform.HMI и SePlatform.Development Studio.



Иерархия элементов - иерархия элементов данных (объекты, поля), к которым могут обращаться исполняемые элементы (формулы , процедуры).

Элементы данных:

- Поле - атомарный элемент данных, хранящий значение некоторого типа;
- Объект - составной элемент данных, который может состоять из нескольких вложенных элементов данных (объектов или полей). Объект может сам хранить значение некоторого типа.

Исполняемые элементы:

- Формула - связана с полем, служит для пересчета значения поля. Пересчет формулы начинается автоматически, как только один из аргументов выражения-формулы меняет свое значение;
- Процедура - связана с объектом, служит для выполнения последовательности инструкций после наступления какого либо события, связанного с объектом.

1.1.1. Относительная адресация

Для работы с иерархией объектов используются следующие ключевые слова и символы:

- me - указатель на текущий объект иерархии;
- @N - обращение к N-му уровню иерархии выше текущего объекта:
  - @0 - текущий объект, аналогично me
  - @1 - родительский объект
  - @2 - родитель родителя

ПРИМЕР

Пусть полный тег объекта: AK.DMN.LU1.NPS1.TANK1.SW1.State.

Тогда:

- me - обращение к самому объекту: AK.DMN.LU1.NPS1.TANK1.SW1.State
- @1 - обращение к родительскому объекту: AK.DMN.LU1.NPS1.TANK1.SW1;
- @2 - "прыжок" на два уровня вверх по иерархии объектов: AK.DMN.LU1.NPS1.TANK1;
- @3 - "прыжок" на три уровня вверх по иерархии объектов: AK.DMN.LU1.NPS1.

$ - символ «доллар» применяется в ситуациях, когда тег объекта противоречит стандартным правилам написания - например, наличие пробелов или запрещенных символов. Символ "$" указывает компилятору, что написанная после него строка (заключенная в двойные кавычки) является путем до объекта в иерархии.

ПРИМЕР

Путь к атрибуту объекта содержит пробелы, поэтому нужно использовать $

```
X: uint1 = $"Уровень 1.Уровень 2.Объект A.Атрибут объекта";
```

1.1.2. Явное чтение аргумента формулы

При компиляции формул для каждого аргумента создаются привязки, которые отслеживают состояния аргументов и запускают пересчет формулы каждый раз при изменении аргумента. Чтобы компилятор не отслеживал состояние некоторого аргумента при пересчете, используйте оператор явного чтения read.

ПРИМЕР

Значение сигнала вычисляется по формуле: **«`X + Y * Z`»**. Необходимо, чтобы значение пересчитывалось при изменениях X и Y и не пересчитывалось при изменениях Z.

```
X + Y * read Z
```

2. Синтаксис

2.1. Алфавит

SePlatform.Om - регистрозависимый язык: переменные названные в разном регистре неэквивалентны друг другу.

Допустимые символы в тексте формул и процедур:

- латинский алфавит (A-Z, a-z);
- русский алфавит (А-Я, а-я);
- цифры (0-9);
- символы:

`	~	!	@	#	%	^	&
*	(	)	_	-	+	=	\
|	{	}	[	]	:	;	"
'	<	>	/	?	.	,	$

2.2. Литералы

Литералы используются в программном коде для обозначения числовых значений, строк, символов или логических значений. Другими словами литерал представляет собой постоянное значение, у которого нет имени.

2.2.1. Логические литералы

Литералы означающие истинность или ложность выражения:

- true
- false

2.2.2. Целочисленные литералы

- 0b<bits> - представление числа в двоичном виде
- 0x<hex> - представление числа

ПРИМЕР

Число 255 в двоичной системе счисления: 0b11111111

Число 255 в шестнадцатеричной системе счисления: 0xFF

2.2.3. Вещественные литералы

Вещественные литералы можно записать:

- В десятичном представлении: X.Y, где X - целая часть числа, а Y - дробная.

Примеры: 2.4, 0.001, 1.0.

Целую часть можно не указывать, в этом случае она считается равной нулю: .1 эквивалентно записи 0.1.

- В экспоненциальном представлении: Me±P, где M - мантиса, ± - знак (может быть не указан, в этом случае считается положительным), P - порядок. Значение вычисляется, как $M \times 10^{\pm P}$.

  Примеры: 1e-2 (равно 0.01), 1e+3 (равно 1000), 314159e-5 (равно 3.14159), 12e3 (эквивалентно 12e+3 и равно 12000).

Тип вещественного литерала (стр. 9) зависит от наличия суффикса:

- без суффикса - тип double
- с суффиксом f - тип float

ПРИМЕР

```
a: double = 0.1; // Значение типа double
b: float = 0.1f; // Значение типа float
```

2.2.4. Строковые литералы

Строковые литералы заключаются в двойные кавычки и могут содержать любые печатные Unicode символы.

Спецсимволы

- \", "" - двойные кавычки
- \n, \r - перевод строки
- \t - символ табуляции
- \\ - обратный слэш

2.3. Ключевые слова

Ключевые слова типов данных:

- int1
- int2
- int4
- int8
- uint1
- uint2
- uint4
- uint8
- float
- double
- bool
- string
- variant
- const
- variant

Логические значения:

- true
- false

Операторы управляющих структур:

- if
- else
- break
- continue

2.4. Комментарии

Комментирование строки - символ //.

```
// Однострочный комментарий
```

Комментирование фрагмента: начало фрагмента /*, конец фрагмента */.

```
/*
Комментирование
нескольких
строк
*/
```

3. Типы данных

Язык SePlatform.Om поддерживает следующие типы данных.

Тип	Описание	Размер	Допустимые значения
int1	Знаковое целое	1 байт	[-128; 127]
uint1	Беззнаковое целое	1 байт	[0; 255]
int2	Знаковое целое	2 байта	[-32 768; 32 767]
uint2	Беззнаковое целое	2 байта	[0; 65 535]
int4	Знаковое целое	4 байта	[-2 147 483 648; 2 147 483 647]
uint4	Беззнаковое целое	4 байта	[0; 4 294 967 295]
int8	Знаковое целое	8 байт	[-9 223 372 036 854 775 808; 9 223 372 036 854 775 807]
uint8	Беззнаковое целое	8 байт	[0; 18 446 744 073 709 551 615]
float	Значение с плавающей запятой	4 байта	[$\pm1.5\times10^{-45}$; $\pm3.4\times10^{38}$] Точность: 6-9 цифр
double	Значение с плавающей запятой	8 байт	[$\pm5.0\times10^{-324}$; $\pm1.7\times10^{308}$] Точность: 15-17 цифр
bool	Логическое значение	1 байт	true, false
string	Текстовая строка в кодировке UTF16	2 байта × кол-во символов Макс. 2 ГБ	
variant	Универсальный тип данных		Принимает значения любых вышеперечисленных элементарных типов

Чтобы компилятор языка автоматически выбрал необходимый тип данных для переменной, используйте ключевое слово var при инициализации значения:

```
W: var = 5.8; // объявление переменной с автоматическим выбором типа
```

Чтобы компилятор языка автоматически выбрал необходимый тип данных для константы, используйте ключевое слово const при инициализации значения:

```
Q: const = 7; // объявление константы с автоматическим выбором типа
X: 5; //краткий синтаксис объявления константы с автоматическим выбором типа
```

3.1. Явное преобразование элементарных типов

Производится с помощью набора функций пространства имен TypeConvert:

- TypeConvert.ToBool
- TypeConvert.ToInt1
- TypeConvert.ToInt2
- TypeConvert.ToInt4
- TypeConvert.ToInt8
- TypeConvert.ToUint1
- TypeConvert.ToUint2
- TypeConvert.ToUint4
- TypeConvert.ToUint8
- TypeConvert.ToFloat
- TypeConvert.ToDouble

ПРИМЕР

Явное преобразования типа (с обрезкой):

```
Num: int1 = TypeConvert.ToInt1(129); // Результат: 127 - максимальное значение для
int1
```

3.2. Неявное преобразование элементарных типов

Неявное преобразование выполняется, когда операция требует один тип данных, а подставляется значение другого типа. В этом случае подставленное значение будет автоматически приведено к нужному типу данных.

Тип	К каким типам может быть приведён
int1	int2, int4, int8, float, double
int2	int4, int8, float, double
int4	int8, float, double
int8	float, double
uint1	int2, int4, int8, uint2, uint4, uint8, float, double
uint2	int4, int8, uint4, uint8, float, double
uint4	int8, uint8, float, double
uint8	float, double
float	double

ОБРАТИТЕ ВНИМАНИЕ

Преобразования:

- int4, uint4 → float
- int8, uint8 → float, double

могут привести к потере точности: итоговое значение может отличаться от оригинального. В остальных случаях преобразование выполняется без потери точности.

ОБРАТИТЕ ВНИМАНИЕ

Неявного преобразования double → float нет. Поэтому выражения следующего вида компилироваться не будут:

```
a: float = 5.4; // 5.4 - вещественный литерал типа double
```

3.3. Тип данных variant

Variant - универсальный тип данных, который может содержать любые виды информации. Способен гибко принимать значения любых элементарных типов:

- целочисленные или вещественные значения
- логические значения
- строковые значения
- неопределенное значение VT_EMPTY.

Применяйте тип данных variant в следующих случаях:

- если при написании скрипта невозможно заранее точно определить тип данных, хранимый в переменной.
- если известно, что содержимое переменной на этапе исполнения скрипта будет меняться в широких диапазонах.

Чтобы создать переменную variant неявным способом, присвойте переменной variant значение любого элементарного типа:

```
v0: variant; //создать пустую переменную типа variant (VT_EMPTY)
v2: variant = 100; //создать переменную типа variant с инициализацией значения
s: string = "строка"; //создать переменную типа string
v1: variant = s; //создать переменную типа variant и присвоить ей значение переменной
```

Также переменную типа variant можно создать явным способом, используя набор функций Variant.From<T>.

3.3.1. Неопределенное значение (VT_EMPTY)

Тип данных variant может принимать неопределенное значение VT_EMPTY в следующих ситуациях:

- не было инициализировано начальное значение типа variant:

```
v0: variant; //создать неинициализированную переменную типа variant
```

- результат операции будет равен VT_EMPTY, если один из аргументов равен VT_EMPTY:

```
v0: variant; //создать неинициализированную переменнуюд variant (VT_EMPTY)
v1: variant = v0 + 1; //v1 = VT_EMPTY, т.к. аргумент v0 = VT_EMPTY
```

- результат операции будет равен VT_EMPTY, если операция логически-некорректна. Например, попытка сложения числа со строкой приведет к значению VT_EMPTY:

```
v0: variant = 1;
v1: variant = "my string";
v2: variant = v0 + v1; //недопустимая операция - сложение числа и строки
```

3.3.2. Набор функций для работы с типом variant

Функции для работы с типом находятся в пространстве имен Variant:

- Variant.From<T> - набор функций для создания переменной типа variant явным способом
- Variant.MayConvertTo<T> - набор функций для проверки возможности конвертации типа variant в какой-либо элементарный тип данных
- Variant.To<T> - набор функций для конвертации типа variant в один из элементарных типов данных
- Variant.Is<T> - набор функций для проверки типа значения в переменной variant

3.3.3. Операции над данными типа variant

- В общем случае диапазон возможных операций определяется типом значения, которое содержится в переменной variant. Например, тип variant, содержащий булево значение (true или false) может быть аргументом в логических операциях.
- На содержимое переменной типа variant не действует ограничение по мощности. Возможно как расширение по мощности, так и сжатие по мощности. Например:

```
int1 = 100;
Y1: int1 = 30;
V1: variant = X1 + Y1;
//переменная V1 будет содержать 130 (расширение до элементарного типа int2)
X2: int2 = 32767;
Y2: int2 = 32762;
V2: variant = X2 - Y2;
//Переменная V2 будет содержать 5 (сжатие до элементарного типа int1)
R1: int1 = X1 + Y1;
//переменная R1 = 127, т.к. достигнут предел мощности типа int1
R2: int2 = X2 - Y2
//переменная R2 = 5, но тип переменной по прежнему int2
```

- Пределы возможных значений ограничены лишь пределами самых мощных типов. Например:

```
X1: uint8 = 18446744073709551615; //максимально-возможное значение uint8
V1: variant = X1 + 1;
//некорректная попытка уместить в переменной значение больше максимального
```

Специальные значения

NaN

NaN (Not-a-Number, не-число) – это особое значение числа с плавающей запятой, которое показывает, что результат операции либо не определён, либо его нельзя представить значением вещественного типа.

При каких условиях возникает NaN:

- Один из аргументов операции равен NaN.
- Один из аргументов операции имеет недопустимое значение.

  Например: извлечение квадратного корня из отрицательного числа.
- Операция не имеет однозначного результата:
  - деление нуля на ноль
  - деление бесконечности на бесконечность
  - умножение нуля на бесконечность
  - сложение положительной и отрицательной бесконечностей

NaN не равен ни одному другому значению, даже самому себе. Чтобы проверить, является ли число значением NaN, используйте функцию Math.IsNaN.

NaN может быть как положительным, так и отрицательным. Чтобы определить знак, используйте функцию Math.SignBit.

Бесконечность

Бесконечность (∞, Infinity) – это особое значение числа с плавающей запятой, которое показывает, что результат операции выходит за границы допустимых значений возвращаемого типа.
Например: 1.0 / 0.
Чтобы проверить, является ли число значением бесконечностью, используйте функцию Math.IsInf.
Бесконечность может быть как положительной (-∞), так и отрицательной (+∞). Чтобы определить знак бесконечности, используйте функцию Math.SignBit.

Отрицательный ноль

Отрицательный ноль (-0) – это значение числа с плавающей запятой, которое может появиться в результате математических операций. Отрицательный ноль равен положительному (обычный ноль) и отличается от него только битом знака. Чтобы определить, является ли ноль положительным или отрицательным, используйте функцию Math.SignBit.

В большинстве случаев знак нуля не влияет на результат выполнения операций. Исключение – операции, результат выполнения которых является знаковым нулём или бесконечностью:

- 1/+0 = +∞; 1/-0 = -∞
- -0*1 = -0; -0/1 = -0

4. Переменные и константы

4.1. Переменные

Переменная - именованное значение, которое может быть изменено в процессе выполнения процедуры. В имени переменной допускается использование символов латинского алфавита (A-Z, a-z), цифр и символа подчеркивания (_).

Возможно объявление инициализированных и неинициализированных переменных:

```
//объявление без инициализации
Идентификатор: ТипДанных;
//объявление с инициализацией
Идентификатор: ТипДанных = Значение;
//объявление с инициализацией
Идентификатор: ТипДанных(Значение);
```

ПРИМЕР

Примеры объявления переменных:

```
X: uint1 = 10; //типизированная переменная с инициализированным значением
T: float; //неинициализированная типизированная переменная
W: var = 5.8; //объявление переменной с автоматическим выбором типа значения
C2: var = 5 + 10; //объявление с вычислением
```

4.2. Константы

Константа - это значение, которое не может быть изменено в процессе выполнения процедуры. Константа может быть именованной или неименованной. В имени константы допускается использование символов латинского алфавита (A-Z, a-z), цифр и символа подчеркивания (_).

```
//Различные способы объявления констант
Q: const = 7; //объявление константы с автоматическим выбором типа
X: 5; //краткий синтаксис объявления константы в автоматическим выбором типа
Y1: uint2 const = 10; // объявление константы с указанием типа
W: 0xF501; //объявление константы - целого числа, заданного в 16-ричном виде
P: 0b00001000; //объявление константы - целого числа, заданного в двоичном виде
M: 1.5; //объявление константы - вещественного числа
N: 1.3e-10; //объявление константы - вещественного числа (экспоненциальный вид)
```

Вычисляемая константа - это константа, полученная путем вычислений из других констант. Может применяться для придания большей семантики коду процедуры.

```
Const1: 5; // объявим одну константу
Const2: 10; // объявим вторую константу
```

```
X: Const1 + Const2; // третья константа как результат сложения двух других
Y: int1 const = X + 10; // типизированная константа как результат вычисления
```

4.3. Область видимости

Область видимости - это часть кода, в рамках которого доступна переменная или константа. Любая объявленная константа или переменная имеет область видимости. Область видимости обозначается фигурными скобками {}. Области видимости могут быть вложенными друг в друга. Константа или переменная, объявленная внутри области видимости не доступна за ними, но доступна во всех вложенных областях видимости.

ПРИМЕР

```
{
  X: var = 10;
  B: bool = true;
  if (B)
  {
    Y: X + 1; // допустимое обращение к X
  }
  else
  {
    //недопустимое обращение Y
    //Y объявлен ни в этой области видимости, ни в родительской
    Y = 0;
  }
}
```

5. Операции

5.1. Присваивание

Присваивание - операция копирования значения и сопутствующей информации из источника в получатель.

```
Dest = Source
```

Dest - получатель значения, Source - источник значения, вычисляемое выражение.

Операция выполняется успешно, если тип значения source приводит к типу значения dest, иначе - ошибка компиляции.

ПРИМЕР

```
res =   X; // Присваивание
res +=  X; // Присваивание со сложением (res = res + X)
res -=  X; // Присваивание с вычитанием (res = res - X)
res *=  X; // Присваивание с умножением (res = res * X)
res /=  X; // Присваивание с делением (res = res / X)
res %=  X; // Присваивание с делением по модулю (res = res % X)
res <<= X; // Присваивание со сдвигом влево (res = res << X)
res >>= X; // Присваивание со сдвигом вправо (res = res >> X)
res |=  X; // Присваивание с дизъюнкцией (res = res | X)
res &=  X; // Присваивание с конъюнкцией (res = res & X)
res ^=  X; // Присваивание с исключающей дизъюнкцией (res = res ^ X)
res = X cmp Y ? resIfTrue : resIfFalse; // Условное присваивание (cmp - это символ
сравнения)
```

5.2. Арифметические операции

Арифметические операции - операции, выполняемые с аргументами числового типа: int1, uint1, int2, uint2, int4, uint4, int8, uint8, float, double:

- отрицание
- сложение
- вычитание
- умножение
- деление
- деление по модулю

Результат операции - значение, имеющее тот же тип данных, что и аргументы. Если аргументы имеют разный тип данных, результат будет иметь тип данных наибольшей мощности из аргументов.

ОБРАТИТЕ ВНИМАНИЕ

Операции отрицание и деление по модулю имеют ограничение на тип аргументов. Допустимые типы аргументов перечислены в описании этих операций.

Отрицание

Унарная операция, меняет знак значения. Ставится перед константой, переменной или скобкой. sОбозначение -.

```
-X
```

Операция применима к следующим типам данных: int1, int2, int4, int8, float, double.

ПРИМЕР

```
X1: int1 = 10;
X2 = -X1; // переменная X2 будет содержать в себе значение -10
```

Сложение

Позволяет выполнять арифметическое сложение двух аргументов. Обознается знаком +.

```
X + Y
```

ПРИМЕР

```
X: int1 = 10;
Y: float = 5.5f;
Z: var = X + Y; // переменная Z будет содержать значение 15.5 (тип данных float)
```

Вычитание

Позволяет выполнять операцию вычитания, результатом которой является новое число, получаемое уменьшением значения первого аргумента на значение второго аргумента. Обознается знаком -.

```
X – Y
```

ПРИМЕР

```
X: int1 = 10;
Y: float = 5.5f;
Z: var = X - Y; // переменная Z будет содержать значение 4.5 (тип данных float)
```

Умножение

Позволяет выполнять арифметическое умножение двух аргументов. Обознается знаком *.

```
X * Y
```

ПРИМЕР

```
X: int1 = 10;
Y: float = 5.5f;
Z: var = X * Y; // переменная Z будет содержать значение 55 (тип данных float)
```

Деление

Позволяет выполнять арифметическое деление двух аргументов. Обознается знаком /.

```
X / Y
```

ПРИМЕР

```
X: int1 = 10;
Y: float = 2.5f;
Z: var = X / Y; // переменная Z будет содержать значение 4 (тип данных float)
```

Деление по модулю

Деление по модулю (деление с остатком) - это арифметическая операция, результатом которой является остаток от деления целого числа на другое целое число.

```
X % Y
```

Операция применима только к целым типам данных: int1, int2, int4, int8, uint1, uint2, uint4, uint8.

ПРИМЕР

```
X: int1 = 10;
Y: int1 = 3;
Z: int1 = X % Y; // переменная Z будет содержать значение 1
```

5.3. Логические операции

Логические операции - операции, выполняемые с аргументами логического типа bool. Результат операции - значение true или false.

ОБРАТИТЕ ВНИМАНИЕ

Чтобы выполнить логическую операцию с аргументами других типов, необходимо выполнить явное приведение этих аргументов к типу bool, иначе произойдёт ошибка компиляции.

В SePlatform.Om доступны следующие логические операции:

- отрицание (инверсия)
- сложение (дизъюнкция)
- умножение (конъюнкция)

Отрицание (инверсия)

Унарная операция. Возвращает true, если аргумент имеет значение false, иначе возвращает false. Обозначение: !

```
!X
```

ПРИМЕР

```
X: bool = true;
Y = !X; // переменная Y будет содержать в себе значение false
```

Сложение (дизъюнкция)

Логическое сложение (дизъюнкция) - это бинарная операция, результат которой равен false, если оба аргумента равны false, иначе - true. Обозначение: ||.

```
X || Y
```

ПРИМЕР

```
Z1: bool = true  || true;  // Результат - true
Z2: bool = true  || false; // Результат - true
Z2: bool = false || true;  // Результат - true
Z3: bool = false || false; // Результат - false
```

Умножение (конъюнкция)

Бинарная операция. Возвращает true, если оба аргумента имеют значение true, иначе возвращает false. Обозначение: &&.

```
X && Y
```

ПРИМЕР

```
Z1: bool = true  && true;  // Результат – true
Z2: bool = true  && false; // Результат – false
Z3: bool = false && true;  // Результат – false
Z4: bool = false && false; // Результат – false
```

5.4. Битовые операции

Битовые операции - операции, производимые над битами аргументов числового типа: uint1, uint2, uint4, uint8 (у некоторых операций есть ограничение на тип аргументов).

В SePlatform.Om доступны следующие битовые операции:

- отрицание (инверсия)
- сложение (дизъюнкция)
- умножение (конъюнкция)
- исключающее сложение
- сдвиг влево
- сдвиг право

Отрицание (инверсия)

Побитовое отрицание (инверсия) - унарная операция, изменяющая значение каждого бита аргумента на противоположное: 0 заменяется на 1, 1 - на 0. Обозначается знаком ~.

```
~X
```

ПРИМЕР

```
X: uint2 = 202; // двоичное значение - 11001010
Y = ~X; // Результат – 53, двоичное значение - 00110101
```

Сложение (дизъюнкция)

Побитовое сложение (дизъюнкция) - бинарная операция, которая побитово сравнивает каждый бит первого аргумента со вторым: в двоичное значение операции запишется 0, если оба бита аргументов равны 0, иначе - 1. Обозначается знаком |.

```
X | Y
```

Умножение (конъюнкция)

Бинарная операция. Каждый бит первого аргумента логически умножается на соответствующий бит второго аргумента. Применяется для типов uint1, uint2, uint4, uint8. Обозначается знаком &.

```
X & Y
```

Исключающее сложение

Бинарная операция. Каждый бит первого аргумента логически складывается с исключением с соответствующим битом второго аргумента. Исключающее сложение - результат равен 0 (false), если оба аргумента равны, иначе - 1 (true). Применяется для типов uint1, uint2, uint4, uint8. Обозначается знаком ^.

```
X ^ Y
```

Сдвиг влево

Бинарная операция. Результат равен сдвигу влево первого аргумента на количество бит, указанных во втором аргументе. Первый аргумент может быть типов - uint1, uint2, uint4, uint8. Второй аргумент - uint1. Обозначение << (альтернативное - shl).

```
X << Y
```

Сдвиг вправо

Бинарная операция. Результат равен сдвигу право первого аргумента на количество бит, указанных во втором аргументе. Первый аргумент может быть типов - uint1, uint2, uint4, uint8. Второй аргумент - uint1. Обозначение >> (альтернативное - shr).

```
X >> Y
```

5.5. Операции со строками

Аргументами в операциях со строками могут являться только строки: тип string.

В SePlatform.Om доступны следующие операции со строками:

- конкатенация
- операции сравнения (описаны ниже)

Сравнение строк выполняется с учётом регистра

Конкатенация

Обозначается знаком +. Результат выполнения операции - строка, являющаяся объединением строк-аргументов.

```
X + Y
```

ПРИМЕР

```
s1: string = "Con";
s2: string = "cat";
s3: string = s1 + s2; // s3 = "Concat"
```

5.6. Операции сравнения

Равно

Бинарная операция. Возвращает true или false. Применяется для всех типов. Обозначение ==.

```
X == Y
```

Не равно

Бинарная операция. Возвращает true или false. Применяется для всех типов. Обозначение !=.

```
X != Y
```

Больше

Бинарная операция. Возвращает true или false. Применяется для всех типов. Обозначение >.

```
X > Y
```

Меньше

Бинарная операция. Возвращает true или false. Применяется для всех типов. Обозначение <.

```
X < Y
```

Больше или равно

Бинарная операция. Возвращает true или false. Применяется для всех типов. Обозначение >=.

```
X >= Y
```

Меньше или равно

Бинарная операция. Возвращает true или false. Применяется для всех типов. Обозначение <=.

```
X <= Y
```

6. Управляющие конструкции

6.1. Условия

Условный оператор if позволяет организовать ветвление кода в зависимости от того, выполняется ли условие.

```
if (condition)
{
  //statements
}
else
{
  //statements
}
```

condition - выражение-условие, вычисляемое в булево значение. Если выражение-условие истинно, то выполняется блок инструкций ветки if, иначе выполняется блок инструкций ветки else. Инструкции if могут быть вложены в другие инструкции if.

ПРИМЕР

```
if (Quality >= 192)
{
  // ветка обработки хорошего качества
  if (condition)
  {
    //statements
  }
  else
  {
  //statements
  }
}
else
{
  // ветка обработки плохого качества
}
```

Стандартная конструкция if может быть заменена более компактной тернарной операцией.

```
<condition>?<statement1>:<statement2>
```

Сначала вычисляется логическое выражение <condition>. Если оно истинно, то вычисляется значение <statement1>, в противном случае - значение <statement2>.

ПРИМЕР

```
MyString: string;
MyString = (Quality >= 192)?("Хорошее качество"):("Плохое качество");
```

6.2. Циклы

Циклы - управляющие конструкции, предназначенные для организации многократного исполнения набора инструкций до тех пор, пока удовлетворяется определенное условие.

Цикл while

Цикл с предварительной проверкой. В отличие от for, цикл while удобно использовать для заранее неизвестного количества итераций.

```
while(condition)
{
  //statement
}
```

Condition - выражение-условие продолжения цикла. Результат выражения должен быть приводим к типу bool. Тело цикла исполняется до тех пор, пока condition равен true. Внутри цикла могут использоваться операторы break (прерывание цикла) и continue (переход на следующую итерацию цикла).

ПРИМЕР

```
//Цикл будет выполняться пока переменная Quality больше или равна 0
while(Quality >= 192)
{
  //statement
}
```

Цикл for

Цикл for обычно используется для выполнения определённого количества итераций.

```
for (init; condition; increment)
{
  //statement
}
```

Здесь:

- init - действие до начала цикла. В большинстве случаев - объявление переменной (счётчика цикла) и присваивание ей начального значения. Объявлять переменную необязательно: присваивать начальное значение можно переменной, объявленной выше.

- condition - условие продолжения цикла. В большинстве случаев - сравнение счётчика цикла с финальным значением.

  Истинность условия проверяется перед каждой итерацией, в том числе перед первой. Если условие не выполняется (возвращает false) - цикл прекращается.

- increment - действие в конце каждой итерации цикла. В большинстве случаев - увеличение счётчика цикла.

ПРИМЕР

```
// Сумма всех чисел от 1 до 1000
sum: int4 = 0;
for (i: int4 = 1; i <= 1000; i += 1)
  sum += i;
```

Внутри цикла можно использовать операторы break (выйти из цикла) и continue (перейти на следующую итерацию цикла).

Любые части init, condition или increment в описании цикла можно опустить.

ОБРАТИТЕ ВНИМАНИЕ

Если не указано условие выхода из цикла, оно принимается равным true. В этом случае для выхода из цикла необходимо использовать оператор break.

ПРИМЕР

Поиск первого числа после 1000, кратного 137.

```
value: int4 = 1000;
for (;; value += 1)
  if (value % 137 == 0)
    break;
// После завершения работы цикла: value = 1096
```

Здесь в описании цикла не указаны:

- init - начальное значение присвоено до цикла.
- condition - условие выхода описано в теле цикла.

7. Функции

7.1. Математические функции

Вызов математических функций осуществляется обращением к пространству имен Math.

ОБРАТИТЕ ВНИМАНИЕ

Если хотя бы один из параметров равен NaN, все функции возвращают NaN, если в описании функции не указано иное.

Функция	Описание
Abs	Возвращает модуль числа.
Acos	Возвращает арккосинус указанного числа.
Acosh	Возвращает обратный гиперболический косинус указанного числа.
Asin	Возвращает арксинус указанного числа.
Asinh	Возвращает обратный гиперболический синус указанного числа.
Atan	Возвращает арктангенс указанного числа.
Atan2	Возвращает арктангенс отношения y/x.
Atanh	Возвращает обратный гиперболический тангенс указанного числа.
Cbrt	Возвращает кубический корень указанного числа.
Ceil	Возвращает число, округлённое до большего целого.
Clamp	Возвращает указанное число, ограниченное дианазоном [min; max].
ClearBit	Возвращает число, у которого обнулён указанный бит.
CopySign	Возвращает число с величиной x и знаком числа y.
Cos	Возвращает косинус угла.
Cosh	Возвращает гиперболический косинус числа.
Dim	Возвращает положительную разность двух числел.
e	Возвращает константу e.
Erf	Возвращает значение функции ошибок.
Erfc	Возвращает значение дополнительной функции ошибок.
Exp	Возвращает число e, возведённое в указанную степень.
Exp2	Возвращает число 2, возведённое в указанную степень.

Функция	Описание
Expm1	Возвращает значение e^x-1.
Floor	Возвращает число, округлённое вниз.
FusedMultiplyAdd	Возвращает умножение-сложение с однократным округлением: (x*y)+z.
Gamma	Возвращает значение гамма-функции для указанного числа.
Hypot	Для указанных длин катетов возвращает длину гипотенузы.
ILogB	Возвращает порядок числа с плавающей запятой в виде знакового целого числа.
IsInf	Проверяет, является ли число бесконечностью (стр. 16).
IsNaN	Проверяет, является ли число значением NaN (стр. 16).
LGamma	Возвращает натуральный логарифм абсолютного значения гамма-функции (стр. 49).
Log	Возвращает натуральный логарифм указанного числа.
Log1p	Возвращает натуральный логарифм числа x+1.
Log2	Возвращает двоичный логарифм числа.
Log10	Возвращает десятичный логарифм числа.
LogB	Возвращает порядок числа с плавающей запятой.
Max	Возвращает большее из двух чисел.
Min	Возвращает меньшее из двух чисел.
NextAfter	Для указанного вещественного числа возвращает ближайшее соседнее число, представимое в рамках типа.
pi	Возвращает число π.
Pow	Возвращает число, возведённое в указанную степень.
Remainder	Возвращает вещественный остаток от деления согласно стандарту IEEE-754.
Round	Возвращает число, округлённое до ближайшего целого или до указанного количества знаков после запятой.
ScaleB	Возвращает произведение числа на 2 в указанной степени: $x*2^n$.
SetBit	Возвращает число, у которого значение указанного бита установлено в true или изменено на указанное значение.
SignBit	Проверяет, является ли число отрицательным.
Sin	Возвращает синус угла.

Функция	Описание
Sinh	Возвращает гиперболический синус числа.
Sqr	Возвращает квадрат числа.
Sqrt	Возвращает квадратный корень числа.
Tan	Возвращает тангенс угла.
Tanh	Возвращает гиперболический тангенс числа.
TestBit	Для числа возвращает значение указанного бита.
ToggleBit	Возвращает число, у которого значение указанного бита изменено на противоположное.
Truncate	Возвращает число, у которого отброшена дробная часть.

7.1.1. Abs

Возвращает модуль числа.

```
uint1 Math.Abs(int1 value)
uint2 Math.Abs(int2 value)
uint4 Math.Abs(int4 value)
uint8 Math.Abs(int8 value)
float Math.Abs(float value)
double Math.Abs(double value)
```

Параметры

Параметр	Тип	Описание
value	int1, int2, int4, int8, float, double	Число, для которого надо вычислить модуль.

Возвращаемое значение

Тип:

- если параметр целочисленного типа - беззнаковый целочисленный тип той же размерности.
- если параметр вещественного типа - такой же, как у параметра.

Модуль числа value.

ПРИМЕР

```
Result: uint1 = Math.Abs(-120); // Результат: 120
```

7.1.2. Acos

Возвращает арккосинус указанного числа.

```
float Math.Acos(float x)
double Math.Acos(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число, являющееся косинусом угла. -1 ≤ x ≤ 1

Возвращаемое значение

Тип - такой же, как у параметра.

Арккосинус x: угол в радианах, косинус которого равен x.

Значение:

- если -1 ≤ x ≤ 1:

0 ≤ результат ≤ π

- иначе:

результат: NaN

ПРИМЕР

```
a: double = Math.Acos(1); // Результат: 0
```

7.1.3. Acosh

Возвращает обратный гиперболический косинус указанного числа.

```
float Math.Acosh(float x)
double Math.Acosh(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число являющееся гиперболическим косинусом угла. 1 ≤ x

Возвращаемое значение

Тип - такой же, как у параметра.

Обратный гиперболический косинус x: угол в радианах, гиперболический косинус которого равен x.

Значение:

› если 1 ≤ x:

0 ≤ результат

› иначе:

результат: NaN

ПРИМЕР

```
a: double = Math.Acosh(1);
// Результат: 0
```

7.1.4. Asin

Возвращает арксинус указанного числа.

```
float Math.Asin(float x)
double Math.Asin(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число, являющееся синусом угла. -1 ≤ x ≤ 1

Возвращаемое значение

Тип - такой же, как у параметра.

Арксинус x: угол в радианах, синус которого равен x.

Значение:

› если -1 ≤ x ≤ 1:

-π/2 ≤ результат ≤ π/2

› иначе:

результат: NaN

ПРИМЕР

```
a: double = Math.Asin(0); // Результат: 0
```

7.1.5. Asinh

Возвращает обратный гиперболический синус указанного числа.

```
float Math.Asinh(float x)
double Math.Asinh(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число, являющееся гиперболическим синусом угла.

Возвращаемое значение

Тип - такой же, как у параметра.

Обратный гиперболический синус x: угол в радианах, гиперболический синус которого равен x.

ПРИМЕР

```
a: double = Math.Asinh(0);
// Результат: 0
```

7.1.6. Atan

Возвращает арктангенс указанного числа.

```
float Math.Atan(float x)
double Math.Atan(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число, являющееся тангенсом угла.

Возвращаемое значение

Тип - такой же, как у параметра.

Арктангенс x - угол в радианах, тангенс которого равен x.

Значение:

- если -∞ < x < +∞:

-π/2 < результат < π/2

- если x = -∞:

результат: -π/2

- если x = +∞:

результат: π/2

ПРИМЕР

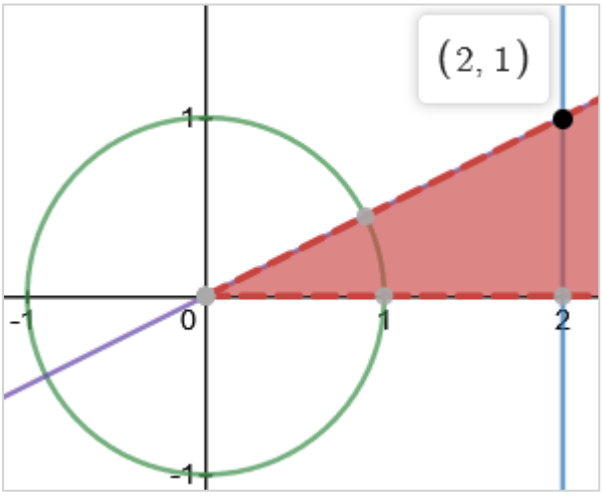
```
a: double = Math.Atan(0); // Результат: 0
```

7.1.7. Atan2

Возвращает арктангенс отношения y/x.

ПРИМЕЧАНИЕ

Геометрический смысл: функция возвращает угол между осью абсцисс и вектором, ведущим из центра координат в точку (x, y).




```
float Math.Atan2(float y, float x)
double Math.Atan2(double y, double x)
```

Параметры

Параметр	Тип	Описание
y	float, double	Ордината точки.
x	float, double	Абсцисса точки. x ≠ 0

Возвращаемое значение

Тип - такой же, как у параметров.

Значение:

- если x, y ≠ 0, x, y ≠ ±∞:

результат - арктангенс отношения y/x: угол в радианах, тангенс которого равен y/x.

-π ≤ результат ≤ π

- если y = ±0, x < 0 или x = -0:

результат: ±π

- если y = ±0, x > 0 или x = +0:

результат: ±0

- если y = ±∞, -∞ < x < +∞:

результат: ±π/2

- если y = ±∞, x = -∞:

результат: ±3π/4

- если y = ±∞, x = +∞:

результат: ±π/4

- если x = ±0, y < 0:

результат: -π/2

- если x = ±0, y > 0:

результат: π/2

- если x = -∞, 0 < y < +∞:

результат: π

- если x = -∞, -∞ < y < 0:

результат: -π

- если x = +∞, 0 < y < +∞:

результат: +0

- если x = +∞, -∞ < y < 0:

результат: -0

ПРИМЕР

```
a: double = Math.Atan2(0, 1);
// Результат: 0
```

7.1.8. Atanh

Возвращает обратный гиперболический тангенс указанного числа.

```
float Math.Atanh(float x)
double Math.Atanh(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число, представляющее гиперболический тангенс. -1 ≤ x ≤ 1

Возвращаемое значение

Тип - такой же, как у параметра.

Обратный гиперболический тангенс x: угол в радианах, гиперболический тангенс которого равен x.

Значение:

- если -1 < x < 1:

-∞ < результат < +∞

- если x = -1:

результат: -∞

- если x = 1:

результат: +∞

- если x < -1 или x > 1:

результат: NaN

7.1.9. Cbrt

Возвращает кубический корень указанного числа.

```
float Math.Cbrt(float x)
double Math.Cbrt(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число, из которого нужно извлечь кубический корень.

Возвращаемое значение

Тип - такой же, как у параметра.

Кубический корень из x.

ПРИМЕР

```
a: double = Math.Cbrt(-125);
// Результат: -5
```

7.1.10. Ceil

Возвращает число, округлённое до большего целого.

```
float Math.Ceil(float value)
double Math.Ceil(double value)
```

Параметры

Параметр	Тип	Описание
value	float, double	Округляемое число.

Возвращаемое значение

Тип - такой же, как у параметра.

Наименьшее целое число, которое больше или равно value.

ПРИМЕР

```
a: double = Math.Ceil(2.11); // Результат: 3
b: double = Math.Ceil(-4.9); // Результат: -4
```

7.1.11. Clamp

Возвращает указанное число, ограниченное дианазоном [min; max].

```
int1 Math.Clamp(int1 value, int1 min, int1 max)
int2 Math.Clamp(int2 value, int2 min, int2 max)
int4 Math.Clamp(int4 value, int4 min, int4 max)
int8 Math.Clamp(int8 value, int8 min, int8 max)
uint1 Math.Clamp(uint1 value, uint1 min, uint1 max)
uint2 Math.Clamp(uint2 value, uint2 min, uint2 max)
```

```
uint4 Math.Clamp(uint4 value, uint4 min, uint4 max)
uint8 Math.Clamp(uint8 value, uint8 min, uint8 max)
float Math.Clamp(float value, float min, float max)
double Math.Clamp(double value, double min, double max)
```

Параметры

Параметр	Тип	Описание
value	int1, int2, int4, int8, uint1, uint2, uint4, uint8, float, double	Ограничиваемое значение.
min	int1, int2, int4, int8, uint1, uint2, uint4, uint8, float, double	Нижняя граница ограничения.
max	int1, int2, int4, int8, uint1, uint2, uint4, uint8, float, double	Верхняя граница ограничения.

Возвращаемое значение

Тип - такой же, как у параметров.

Число value, ограниченное диапазоном [min; max]:

- если min ≤ value ≤ max:

результат: value

- если value < min:

результат: min

- если value > max:

результат: max

ПРИМЕР

```
a: int4 = Math.Clamp(5, 0, 10); // Результат: 5
b: int4 = Math.Clamp(15, 0, 10); // Результат: 10
c: int4 = Math.Clamp(-5, 0, 10); // Результат: 0
```

7.1.12. ClearBit

Возвращает число, у которого обнулён указанный бит.

```
uint1 Math.ClearBit(uint1 value, uint1 n)
uint2 Math.ClearBit(uint2 value, uint1 n)
uint4 Math.ClearBit(uint4 value, uint1 n)
uint8 Math.ClearBit(uint8 value, uint1 n)
```

Параметры

Параметр	Тип	Описание
value	uint1, uint2, uint4, uint8	Число, над которым будет выполняться функция.
n	uint1	Номер бита, нумерация начинается с нуля.

Возвращаемое значение

Тип - такой же, как у параметра value.

Значение параметра value, у которого обнулён n-й бит. Если n больше размерности типа value, возвращает значение value без изменений.

ПРИМЕР

У числа 30 (11110 в двоичной записи) обнулить бит под номером 1:

```
a: uint4 = Math.ClearBit(30, 1) // Результат: 28 (11100 в двоичной записи).
```

7.1.13. CopySign

Возвращает число с величиной x и знаком числа y.

```
float Math.CopySign(float x, float y)
double Math.CopySign(double x, double y)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число, величина которого используется в возвращаемом значении.
y	float, double	Число, знак которого используется в возвращаемом значении.

Возвращаемое значение

Тип - такой же, как у параметров.

Величина x со знаком, как у y.

ПРИМЕР

```
a: double = Math.CopySign(5, 1); // Результат: 5
b: double = Math.CopySign(5, -3); // Результат: -5
c: double = Math.CopySign(-5, 10); // Результат: 5
d: double = Math.CopySign(-5, -4); // Результат: -5
```

7.1.14. Cos

Возвращает косинус угла.

```
float Math.Cos(float value)
double Math.Cos(double value)
```

Параметры

Параметр	Тип	Описание
value	float, double	Угол в радианах.

ПРИМЕЧАНИЕ

Чтобы преобразовать градусы в радианы, можете использовать следующую формулу:

```
radians = degrees * Math.pi()/180
```

Возвращаемое значение

Тип - такой же, как у параметра.

Косинус угла value.

ПРИМЕР

```
Result: double = Math.Cos(1); // Результат: 0.54030230586814
```

7.1.15. Cosh

Возвращает гиперболический косинус числа.

$$ch(x) = \frac{e^{x}+e^{-x}}{2}$$

```
float Math.Cosh(float x)
double Math.Cosh(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Угол в радианах.

Возвращаемое значение

Тип - такой же, как у параметра.

Гиперболический косинус x:

Значение:

- если -∞ < x < +∞

1 ≤ результат

- если x = ±∞:

результат: +∞

ПРИМЕР

```
a: double = Math.Cosh(0); // Результат: 1
```

7.1.16. Dim

Возвращает положительную разность двух числел.

```
float Math.Dim(float x, float y)
double Math.Dim(double x, double y)
```

Параметры

Параметр	Тип	Описание
x	float, double	Уменьшаемое.
y	float, double	Вычитаемое.

Возвращаемое значение

Тип - такой же, как у параметров.

Значение:

- если x≥y:

результат: x-y

- если x < y:

результат: 0

ПРИМЕЧАНИЕ

Функция аналогична функции Math.Max(x-y, 0) и отличается от неё тем, что возвращает NaN, если один из аргументов равен NaN.

ПРИМЕР

```
a: double = Math.Dim(5, 1); // Результат: 4
b: double = Math.Dim(5, 6); // Результат: 0
```

7.1.17. e

Возвращает константу e.

```
double Math.e()
```

Возвращаемое значение

Тип - double.

Число 2.7182818284590451.

ПРИМЕР

```
e: double = Math.e(); // Результат: 2.7182818284590451
```

7.1.18. Erf

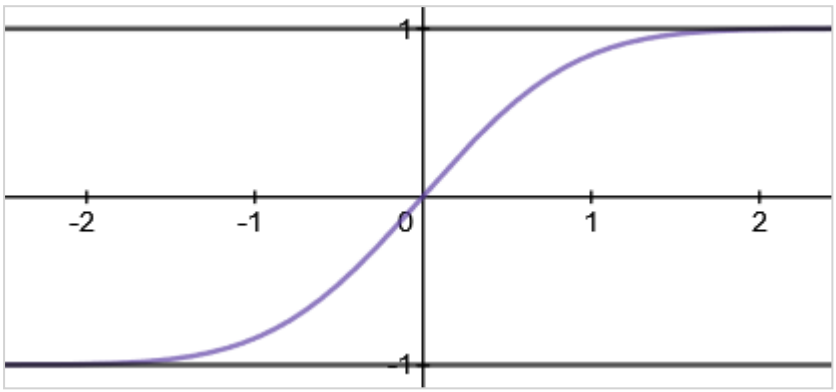
Возвращает значение функции ошибок.

ПРИМЕЧАНИЕ

Функция ошибок - неэлементарная функция в теории вероятности и статистике, которая вычисляется по формуле:

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

График функции:




```
float Math.Erf(float x)
double Math.Erf(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Аргумент функции

Возвращаемое значение

Тип - такой же, как у параметра.

Значение:

- если -∞ < x < +∞:

результат - значение функции ошибок для x.

-1 < результат < 1

- если x = -∞:

результат: -1

- если x = +∞:

результат: 1

7.1.19. Erfc

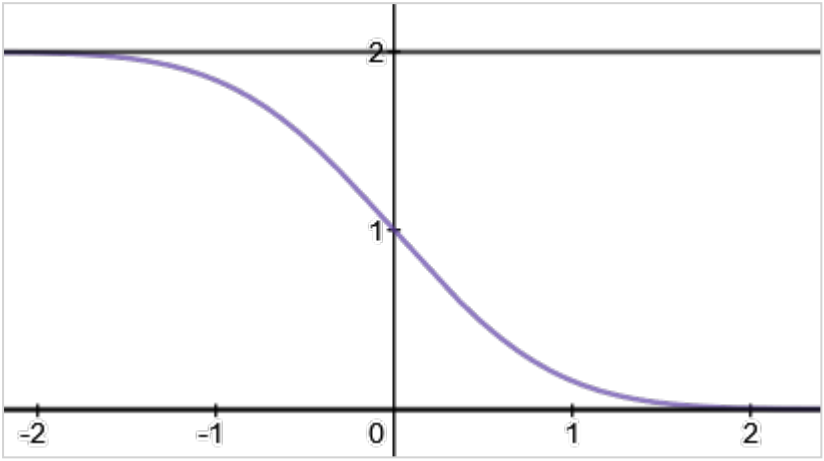
Возвращает значение дополнительной функции ошибок.

ПРИМЕЧАНИЕ

Дополнительная функция ошибок определяется через функцию ошибок по формуле:

$$erfc(x) = 1 - erf(x) = \frac{2}{\sqrt{\pi}} \int_{x}^{\infty} e^{-t^2} dt$$

График функции:




```
float Math.Erfc(float x)
double Math.Erfc(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Аргумент функции.

Возвращаемое значение

Тип - такой же, как у параметра.

Значение:

- если -∞ < x < +∞:

результат - значение обратной функции ошибок для x.

0 < результат < 2

- если x = -∞:

результат: 2

- если x = +∞:

результат: 0

7.1.20. Exp

Возвращает число e, возведённое в указанную степень.

```
float Math.Exp(float p)
double Math.Exp(double p)
```

Параметры

Параметр	Тип	Описание
p	float, double	Показатель степени.

Возвращаемое значение

Тип - такой же, как у параметра.

Число e, возведённое в степень p.

ПРИМЕР

Возвести число e в степень 2.5:

```
Result: double = Math.Exp(2.5); // Результат: 12.1824939607035
```

7.1.21. Exp2

Возвращает число 2, возведённое в указанную степень.

```
float Math.Exp2(float x)
double Math.Exp2(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Показатель степени.

Возвращаемое значение

Тип - такой же, как у параметра.

Число 2 в степени x.

ПРИМЕР

```
a: double = Math.Exp2(5); // Результат: 32
b: double = Math.Exp2(0); // Результат: 1
c: double = Math.Exp2(-1); // Результат: 0.5
d: double = Math.Exp2(5); // Результат: 32
```

7.1.22. Expm1

Возвращает значение e^x-1.

```
float Math.Expm1(float x)
double Math.Expm1(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Показатель степени.

Возвращаемое значение

Тип - такой же, как у параметра.

Значение e^x-1.

Значение:

- если -∞ < x < +∞:

результат - значение e^x-1.

-1 < результат < +∞

- если x = -∞:

результат: -1

- если x = +∞:

результат: +∞

7.1.23. Floor

Возвращает число, округлённое вниз.

```
float Math.Floor(float value)
double Math.Floor(double value)
```

Параметры

Параметр	Тип	Описание
value	float, double	Округляемое число.

Возвращаемое значение

Тип - такой же, как у параметра.

Наибольшее целое число, которое меньше или равно value.

ПРИМЕР

```
a: double = Math.Floor(7.56); // Результат: 7
b: double = Math.Floor(-5.1); // Результат: -6
```

7.1.24. FusedMultiplyAdd

Возвращает умножение-сложение с однократным округлением: (x*y)+z..

ПРИМЕЧАНИЕ

Вычисление выполняется быстрее и более точно, чем при выполнении тех же операций по-отдельности, т.к. выполняется через аппаратные инструкции процессора.

```
float Math.FusedMultiplyAdd(float x, float y, float z)
double Math.FusedMultiplyAdd(double x, double y, double z)
```

Параметры

Параметр	Тип	Описание
x	float, double	Первый множитель.
y	float, double	Второй множитель.
z	float, double	Слагаемое.

Возвращаемое значение

Тип - такой же, как у параметров.

Результат вычисления (x*y)+z.

7.1.25. Gamma

Возвращает значение гамма-функции для указанного числа.

ПРИМЕЧАНИЕ

Описание гамма-функции: https://ru.dsplib.org/content/gamma_func/gamma_func.html

```
float Math.Gamma(float x)
double Math.Gamma(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Аргумент функции.

Возвращаемое значение

Тип - такой же, как у параметра.

Значение гамма-функции для x.

7.1.26. Hypot

Для указанных длин катетов возвращает длину гипотенузы.

```
float Math.Hypot(float x, float y)
double Math.Hypot(double x, double y)
```

Параметры

Параметр	Тип	Описание
x	float, double	Длина первого катета. Может иметь отрицательное значение, т.к. при рассчёте возводится в квадрат.
y	float, double	Длина второго катета. Может иметь отрицательное значение, т.к. при рассчёте возводится в квадрат.

Возвращаемое значение

Тип - такой же, как у параметров.

Длина гипотенузы: корень из суммы квадратов x и y:

$$result = \sqrt{x^2 + y^2}$$

0 ≤ результат

7.1.27. ILogB

Возвращает порядок числа с плавающей запятой в виде знакового целого числа. Порядок – целое число, которое задаёт нужную степень двойки. Обычно это не истинная величина порядка, а сдвинутая на некоторую константу таким образом, чтобы число было неотрицательным. Так, наименьший возможный порядок (он отрицательный) представлен числом 0.

```
int4 Math.ILogB(float x)
int4 Math.ILogB(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число.

Возвращаемое значение

Тип - int4.

Порядок числа x в виде знакового целого.

7.1.28. IsInf

Проверяет, является ли число бесконечностью (стр. 16).

```
bool Math.IsInf(float x)
bool Math.IsInf(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число.

Возвращаемое значение

Тип - bool.

Значение:

- если x = ±∞:

результат: true

- иначе:

результат: false

7.1.29. IsNaN

Проверяет, является ли число значением NaN (стр. 16).

```
bool Math.IsNaN(float x)
bool Math.IsNaN(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число.

Возвращаемое значение

Тип - bool.

Значение:

- если x = NaN:

результат: true

- иначе:

результат: false

7.1.30. LGamma

Возвращает натуральный логарифм абсолютного значения гамма-функции (стр. 49).

```
float Math.LGamma(float x)
double Math.LGamma(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Аргумент функции.

Возвращаемое значение

Тип - такой же, как у параметра.

Возвращает натуральный логарифм значения гамма-функции для x.

7.1.31. Log

Возвращает натуральный логарифм указанного числа.

```
float Math.Log(float x)
float Math.Log(float x, float base)
double Math.Log(double x)
double Math.Log(double x, float base)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число, натуральный логарифм которого нужно найти. 0 ≤ x
base	float, double	Опциональный параметр. Основание логарифма. Если не указан, используется значение e: берётся натуральный логарифм числа. 0 < base; base ≠ 1

Возвращаемое значение

Тип - такой же, как у параметра.

Значение:

- если base не указан:

результат - натуральный логарифм x: степень, в которую надо возвести число e, чтобы получить x.

- если base указан:

результат - логарифм x по основанию base: степень, в которую надо возвести base, чтобы получить x.

-∞ < результат < +∞

7.1.32. Log1p

Возвращает натуральный логарифм числа x+1.

ПРИМЕЧАНИЕ

Функция возвращает более точный результат, чем Math.Log(x+1) для значений x близких к 0.

```
float Math.Log1p(float x)
double Math.Log1p(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число.

Возвращаемое значение

Тип - такой же, как у параметра.

Натуральный логарифм числа x+1: логарифм числа x+1 по основанию e.

-∞ < результат < +∞

7.1.33. Log2

Возвращает двоичный логарифм числа.

```
float Math.Log2(float x)
double Math.Log2(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число, двоичный логарифм которого нужно найти. 0 < x

Возвращаемое значение

Тип - такой же, как у параметра.

Значение:

- если x > 0:

результат - двоичный логарифм x: степень, в которую нужно возвести 2, чтобы получить x.

- если x = 0:

результат: -∞

- если x = +∞:

результат: +∞

- если x < 0:

результат: NaN

ПРИМЕР

```
a: double = Math.Log2(2); // Результат: 1
b: double = Math.Log2(16); // Результат: 4
c: double = Math.Log2(1); // Результат: 0
d: double = Math.Log2(0.5); // Результат: -1
```

7.1.34. Log10

Возвращает десятичный логарифм числа.

```
float Math.Log10(float x)
double Math.Log10(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число, десятичный логарифм которого нужно найти. 0 < x

Возвращаемое значение

Тип - такой же, как у параметра.

Значение:

- если x > 0:

результат - десятичный логарифм x: степень, в которую нужно возвести 10, чтобы получить x.

- если x = 0:

результат: -∞

- если x = +∞:

результат: +∞

- если x < 0:

результат: NaN

ПРИМЕР

```
a: double = Math.Log10(10); // Результат: 1
b: double = Math.Log10(100); // Результат: 2
c: double = Math.Log10(1); // Результат: 0
d: double = Math.Log10(0.1); // Результат: -1
```

7.1.35. LogB

Возвращает порядок числа с плавающей запятой. Порядок – целое число, которое задаёт нужную степень двойки. Обычно это не истинная величина порядка, а сдвинутая на некоторую константу таким образом, чтобы число было неотрицательным. Так, наименьший возможный порядок (он отрицательный) представлен числом 0.

```
float Math.LogB(float x)
double Math.LogB(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число.

Возвращаемое значение

Тип - такой же, как у параметра.

Значение:

- если x ≠ 0 и x ≠ ±∞:

результат - порядок числа x в виде знакового числа с плавающей запятой.

- если x = 0:

результат: -∞

- если x = ±∞:

результат: +∞

7.1.36. Max

Возвращает большее из двух чисел.

```
int1 Math.Max(int1 X, int1 Y)
int2 Math.Max(int2 X, int2 Y)
int4 Math.Max(int4 X, int4 Y)
int8 Math.Max(int8 X, int8 Y)
uint1 Math.Max(uint1 X, uint1 Y)
uint2 Math.Max(uint2 X, uint2 Y)
uint4 Math.Max(uint4 X, uint4 Y)
uint8 Math.Max(uint8 X, uint8 Y)
float Math.Max(float X, float Y)
double Math.Max(double X, double Y)
```

Параметры

Параметр	Тип	Описание
X	int1, int2, int4, int8, uint1, uint2, uint4, uint8, float, double	Первое число.
Y	int1, int2, int4, int8, uint1, uint2, uint4, uint8, float, double	Второе число.

Возвращаемое значение

Тип - совпадает с типом параметров.

Большее из чисел X и Y.

ПРИМЕР

```
result: uint1 = Math.Max(43, 47); // Результат: 47
```

7.1.37. Min

Возвращает меньшее из двух чисел.

```
int1 Math.Min(int1 X, int1 Y)
int2 Math.Min(int2 X, int2 Y)
int4 Math.Min(int4 X, int4 Y)
int8 Math.Min(int8 X, int8 Y)
uint1 Math.Min(uint1 X, uint1 Y)
uint2 Math.Min(uint2 X, uint2 Y)
uint4 Math.Min(uint4 X, uint4 Y)
uint8 Math.Min(uint8 X, uint8 Y)
float Math.Min(float X, float Y)
double Math.Min(double X, double Y)
```

Параметры

Параметр	Тип	Описание
X	int1, int2, int4, int8, uint1, uint2, uint4, uint8, float, double	Первое число.
Y	int1, int2, int4, int8, uint1, uint2, uint4, uint8, float, double	Второе число.

Возвращаемое значение

Тип - совпадает с типом параметров.

Меньшее из чисел X и Y.

ПРИМЕР

```
result: uint1 = Math.Min(43, 47); // Результат: 43
```

7.1.38. NextAfter

Для указанного вещественного числа возвращает ближайшее соседнее число, представимое в рамках типа.

```
float Math.NextAfter(float from, float to)
double Math.NextAfter(double from, float to)
```

Параметры

Параметр	Тип	Описание
from	float, double	Число, для которого будет искаться сосед.

Параметр	Тип	Описание
to	float, double	Число, по направлению к которому будет искаться сосед.

Возвращаемое значение

Тип - такой же, как у параметров.

Число, ближайшее к from в направлении to и представимое в рамка типа.

7.1.39. pi

Возвращает число π.

```
double Math.pi()
```

Возвращаемое значение

Тип - double.

Число 3.1415926535897931.

ПРИМЕР

```
Result: double = Math.pi(); // Результат: 3.1415926535897931
```

7.1.40. Pow

Возвращает число, возведённое в указанную степень.

```
float Math.Pow(float X, float Y)
double Math.Pow(double X, double Y)
```

Параметры

Параметр	Тип	Описание
X	float, double	Первое число.
Y	float, double	Второе число.

Возвращаемое значение

Тип - совпадает с типом параметров.

Число X, возведённое в степень Y.

ПРИМЕР

Возвести число 1,5 в степень 4:

```
Result: double = Math.Pow(1.5, 4); // Результат: 5.0625
```

7.1.41. Remainder

Возвращает вещественный остаток от деления согласно стандарту IEEE-754.

```
float Math.Remainder(float x, float y)
double Math.Remainder(double x, float y)
```

Параметры

Параметр	Тип	Описание
x	float, double	Делимое.
y	float, double	Делитель.

Возвращаемое значение

Тип - такой же, как у параметров.

Возвращает остаток от деления x/y согласно определению в стандарте IEEE-754: $result = x - (y * Q)$, где Q - частное x/y, округлённое до ближайшего целого.

7.1.42. Round

Возвращает число, округлённое до ближайшего целого или до указанного количества знаков после запятой.

```
float Math.Round(float value)
float Math.Round(float value, uint4 precision)
double Math.Round(double value)
double Math.Round(double value, uint4 precision)
```

Параметры

Параметр	Тип	Описание
value	float, double	Округляемое число.

Параметр	Тип	Описание
precision	uint4	Опциональный параметр. Количество знаков после запятой. Если не указан, принимается равным нулю (округление до целого).

Возвращаемое значение

Тип - такой же, как у параметра value.

Число value, округлённое до ближайшего целого или до указанного количества знаков после запятой.

ПРИМЕР

Округлить до целого:

```
a: double = Math.Round(1.1); // Результат: 1
b: double = Math.Round(7.5); // Результат: 8
c: double = Math.Round(-1.5); // Результат: -2
```

ПРИМЕР

Округлить до указанного количества знаков после запятой:

```
x: double = Math.Round(4.6666666, 2); // Результат: 4.67
my_pi: double = Math.Round(Math.pi(), 5); // Результат: 3,14159
```

7.1.43. ScaleB

Возвращает произведение числа на 2 в указанной степени: $x*2^n$.

ПРИМЕЧАНИЕ

Вычисление выполняется быстрее, чем при выполнении тех же операций по-отдельности, т.к. выполняется через аппаратные инструкции процессора.

```
float Math.ScaleB(float x, int4 exp)
double Math.ScaleB(double x, int4 exp)
```

Параметры

Параметр	Тип	Описание
x	float, double	Множитель.
exp	int4	Показатель степени, в которую надо возвести 2.

Возвращаемое значение

Тип - такой же, как у параметра x.

Значение $x*2^{exp}$.

7.1.44. SetBit

Возвращает число, у которого значение указанного бита установлено в true или изменено на указанное значение.

```
uint1 Math.SetBit(uint1 value, uint1 n)
uint2 Math.SetBit(uint2 value, uint1 n)
uint4 Math.SetBit(uint4 value, uint1 n)
uint8 Math.SetBit(uint8 value, uint1 n)
uint1 Math.SetBit(uint1 value, uint1 n, bool bitValue)
uint2 Math.SetBit(uint2 value, uint1 n, bool BitValue)
uint4 Math.SetBit(uint4 value, uint1 n, bool bitValue)
uint8 Math.SetBit(uint8 value, uint1 n, bool BitValue)
```

Параметры

Параметр	Тип	Описание
value	uint1, uint2, uint4, uint8	Число, над которым будет выполняться функция.
n	uint1	Номер бита, нумерация начинается с нуля.
bitValue	bool	(опциональный параметр) Новое значение n-го бита: true (1) или false (0). Если не указан, значение бита будет установлено в true.

Возвращаемое значение

Тип - такой же, как у параметра value.

Значение параметра value, у которого n-му биту установлено значение bitValue или true, если bitValue не указан. Если n больше размерности типа value, возвращает значение value без изменений.

ПРИМЕР

У числа 22 (10110 в двоичной записи) установить бит под номером 3:

```
a: uint4 = Math.SetBit(22, 3) // Результат: 30 (11100 в двоичной записи).
```

У числа 22 (10110 в двоичной записи) изменить бит под номером 5:

```
b: uint4 = Math.SetBit(22, 5) // Результат: 54 (110100 в двоичной записи).
```

У числа 19 (10011 в двоичной записи), обнулить значение бита под номером 0:

```
c: uint4 = Math.SetBit(19, 0, false) // Результат: 18 (10010 в двоичной записи).
```

7.1.45. SignBit

Проверяет, является ли число отрицательным.

```
bool Math.SignBit(float x)
bool Math.SignBit(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Число, знак которого проверяем.

Возвращаемое значение

Тип - bool.

Значение:

- если x < 0, x = -0, x = -∞ или x - отрицательный NaN:

результат: true

- если x > 0, x = +0, x = +∞ или x - положительный NaN:

результат: false

7.1.46. Sin

Возвращает синус угла.

```
float Math.Sin(float value)
double Math.Sin(double value)
```

Параметры

Параметр	Тип	Описание
value	float, double	Угол в радианах.

ПРИМЕЧАНИЕ

Чтобы преобразовать градусы в радианы, можете использовать следующую формулу:

```
radians = degrees * Math.pi()/180
```

Возвращаемое значение

Тип - такой же, как у параметра.

Синус угла value.

ПРИМЕР

```
Result: double = Math.Sin(1); // Результат: 0.841470984807897
```

7.1.47. Sinh

Возвращает гиперболический синус числа.

$$sh(x) = \frac{e^x - e^{-x}}{2}$$

```
float Math.Sinh(float x)
double Math.Sinh(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Угол в радианах.

Возвращаемое значение

Тип - такой же, как у параметра.

Значение:

- если -∞ < x < +∞:

результат - гиперболический синус x.

- если x = -∞:

результат: -∞

- если x = +∞:

результат: +∞

ПРИМЕР

```
a: double = Math.Sinh(0); // Результат: 0
```

7.1.48. Sqr

Возвращает квадрат числа.

```
int1 Math.Sqr(int1 value)
int2 Math.Sqr(int2 value)
int4 Math.Sqr(int4 value)
int8 Math.Sqr(int8 value)
uint1 Math.Sqr(uint1 value)
uint2 Math.Sqr(uint2 value)
uint4 Math.Sqr(uint4 value)
uint8 Math.Sqr(uint8 value)
double Math.Sqr(double value)
```

Параметры

Параметр	Тип	Описание
value	int1, int2, int4, int8, uint1, uint2, uint4, uint8, double	Возводимое в квадрат число.

Возвращаемое значение

Тип - такой же, как у параметра.

Число value, возведённое в квадрат.

ПРИМЕР

```
Result: int4 = Math.Sqr(12); // Результат: 144
```

7.1.49. Sqrt

Возвращает квадратный корень числа.

```
float Math.Sqrt(float value)
double Math.Sqrt(double value)
```

Параметры

Параметр	Тип	Описание
value	float, double	Число, квадратный корень которого нужно найти. Должно быть неотрицательным.

Возвращаемое значение

Тип - такой же, как у параметра.

Значение:

- если x ≥ 0:

результат - квадратный корень из числа value.

- если x < 0:

результат: NaN

ПРИМЕР

```
a: double = Math.Sqrt(1.69); // Результат: 1.3
```

7.1.50. Tan

Возвращает тангенс угла.

```
float Math.Tan(float x)
double Math.Tan(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Угол в радианах.

ПРИМЕЧАНИЕ

Чтобы преобразовать градусы в радианы, можете использовать следующую формулу:

```
radians = degrees * Math.pi()/180
```

Возвращаемое значение

Тип - такой же, как у параметра x.

Значение:

- если -∞ < x < +∞:

результат - тангенс угла x.

› если x = ±∞:

результат: NaN

ПРИМЕР

```
a: double = Math.Tan(0); // Результат: 0
b: double = Math.Tan(Math.pi()/4); Результат: 1
```

7.1.51. Tanh

Возвращает гиперболический тангенс числа.

$$th(x) = \frac{sh(x)}{ch(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

```
float Math.Tanh(float x)
double Math.Tanh(double x)
```

Параметры

Параметр	Тип	Описание
x	float, double	Угол в радианах.

Возвращаемое значение

Тип - такой же, как у параметра.

Значение:

› если -∞ < x < +∞:

результат - гиперболический тангенс x.

-1 < результат < 1

› если x = -∞:

результат: -1

› если x = +∞:

результат: 1

7.1.52. TestBit

Для числа возвращает значение указанного бита.

Проверить бит X. Возвращает true, если в бите с номером bitNo установлена единица, иначе false. Если bitNo больше размерности типа X, возвращает false.

```
bool Math.TestBit(uint1 value, uint1 bitNo)
bool Math.TestBit(uint2 value, uint1 bitNo)
bool Math.TestBit(uint4 value, uint1 bitNo)
bool Math.TestBit(uint8 value, uint1 bitNo)
```

Параметры

Параметр	Тип	Описание
value	uint1, uint2, uint4, uint8	Число, над которым будет выполняться функция.
bitNo	uint1	Номер бита, нумерация начинается с нуля.

Возвращаемое значение

Тип - bool.

Значение бита под номером bitNo в числе value.

ПРИМЕР

Для числа 56 (111000 в двоичной записи) определить значение битов:

```
a: bool = Math.TestBit(56, 0); // Результат: false
b: bool = Math.TestBit(56, 3); // Результат: true
c: bool = Math.TestBit(56, 6); // Результат: false
```

7.1.53. ToggleBit

Возвращает число, у которого значение указанного бита изменено на противоположное.

```
uint1 Math.ToggleBit(uint1 value, uint1 n)
uint2 Math.ToggleBit(uint2 value, uint1 n)
uint4 Math.ToggleBit(uint4 value, uint1 n)
uint8 Math.ToggleBit(uint8 value, uint1 n)
```

Параметры

Параметр	Тип	Описание
value	uint1, uint2, uint4, uint8	Число, над которым будет выполняться функция.
n	uint1	Номер бита, начиная с нуля.

Возвращаемое значение

Тип - такой же, как у параметра value.

Значение параметра value, у которого значение n-го бита изменено на противоположное.

ПРИМЕР

У числа 22 (10110 в двоичной записи) изменить бит под номером 3:

```
a: uint1 = Math.ToggleBit(22, 3) // Результат: 30 (11100 в двоичной записи).
```

У числа 22 (10110 в двоичной записи) изменить бит под номером 1:

```
b: uint1 = Math.ToggleBit(22, 1) // Результат: 20 (10100 в двоичной записи).
```

У числа 22 (10110 в двоичной записи) изменить бит под номером 5:

```
b: uint1 = Math.ToggleBit(22, 5) // Результат: 54 (110110 в двоичной записи).
```

7.1.54. Truncate

Возвращает число, у которого отброшена дробная часть.

```
float Math.Truncate(float value)
double Math.Truncate(double value)
```

Параметры

Параметр	Тип	Описание
value	float, double	Усекаемое число.

Возвращаемое значение

Тип - такой же, как у параметра.

Целая часть числа value.

ПРИМЕР

```
a: double = Math.Truncate(18.7585); // Результат: 18
a: double = Math.Truncate(-7.2); // Результат: -7
```

7.2. Строковые функции

Вызов строковых функций осуществляется обращением к пространству имен String или Str.

ОБРАТИТЕ ВНИМАНИЕ

Пространство имён Str является устаревшим и оставлено для обратной совместимости. Рекомендуем обращаться к String.

Функция	Описание
Add	Возвращает объединение двух строк.
Concat	Возвращает объединение двух строк.
Contains	Проверяет, есть ли в строке указанная подстрока.
EndsWith	Проверяет, заканчивается ли строка на указанную подстроку.
EQ	Лексикографически сравнивает две строки аналогично оператору "равно".
Equals	Проверяет две строки на равенство.
GE	Лексикографически сравнивает две строки аналогично оператору "больше или равно".
GT	Лексикографически сравнивает две строки аналогично оператору "больше".
IndexOf	Возвращает индекс первого вхождения подстроки в указанную строку.
Insert	Возвращает строку, в указанное место которой вставлена другая строка.
IsValidFormat	Проверяет, является ли указанная строка корректной строкой формата printf.
LastIndexOf	Возвращает индекс последнего вхождения подстроки в указанную строку.
LE	Лексикографически сравнивает две строки аналогично оператору "меньше или равно".
Length	Возвращает количество символов в строке.
LT	Лексикографически сравнивает две строки аналогично оператору "меньше".
NE	Лексикографически сравнивает две строки аналогично оператору "не равно".
Remove	Возвращает строку, в которой начиная с указанной позиции удалено указанное количество символов.
Replace	Возвращает строку, в которой все вхождения одной подстроки заменены на другую подстроку.
Reserve	Возвращает пустую строку с местом, зарезервированным для хранения указанного количества символов.
StartsWith	Проверяет, начинается ли строка с указанной подстроки.
SubString	Для указанной строки возвращает подстроку.
ToBool	Конвертирует указанную строку в значение типа bool.
ToDouble	Конвертирует указанную строку в число типа double.

Функция	Описание
ToFloat	Конвертирует указанную строку в число типа float.
ToInt1	Конвертирует указанную строку в число типа int1.
ToInt2	Конвертирует указанную строку в число типа int2.
ToInt4	Конвертирует указанную строку в число типа int4.
ToInt8	Конвертирует указанную строку в число типа int8.
ToLocalizedString	Возвращает строковое представление указанного значения с учётом системного языка.
ToLower	Возвращает строку, в которой все символы приведены к нижнему регистру.
ToString	Возвращает строковое представление указанного значения.
ToUint1	Конвертирует указанную строку в число типа uint1.
ToUint2	Конвертирует указанную строку в число типа uint2.
ToUint4	Конвертирует указанную строку в число типа uint4.
ToUint8	Конвертирует указанную строку в число типа uint8.
ToUpper	Возвращает строку, в которой все символы приведены к верхнему регистру.
Trim	Возвращает строку, в которой удалены пробелы в начале и в конце строки.

7.2.1. Add

Возвращает объединение двух строк.

```
string String.Add(string x, string y)
```

Функция является полным аналогом функции String.Concat.

7.2.2. Concat

Возвращает объединение двух строк.

```
string String.Concat(string x, string y)
```

Параметры

Параметр	Тип	Описание
x	string	Первая строка.

Параметр	Тип	Описание
y	string	Вторая строка.

Возвращаемое значение

Тип - string.

Строка x, в конец которой добавлена строка y.

ПРИМЕР

```
a: string = String.Concat("Note: ", "Message");
// Результат: "Note: Message"
```

ПРИМЕЧАНИЕ

Функция аналогична сложению двух строк: s1 + s2.

7.2.3. Contains

Проверяет, есть ли в строке указанная подстрока.

```
bool String.Contains(string x, string y)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, в которой будем проверять наличие подстроки.
y	string	Подстрока, которую будем искать.

Возвращаемое значение

Тип - bool.

true, если в строке x есть подстрока y, false - если подстроки нет. Если y - пустая подстрока, то возвращается true.

ПРИМЕР

```
a: bool = String.Contains("Hello, World!", "World"); // Результат: true
b: bool = String.Contains("Hello, World!", "hello"); // Результат: false
c: bool = String.Contains("Hello, World!", "");      // Результат: true
```

7.2.4. EndsWith

Проверяет, заканчивается ли строка на указанную подстроку.

```
bool String.EndsWith(string x, string y)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем проверять.
y	string	Подстрока, которую будем искать в конце строки.

Возвращаемое значение

Тип - bool.

true, если строка x заканчивается на подстроку y; иначе - false. Если y - пустая строка, вернётся true.

ПРИМЕР

```
a: bool = String.EndsWith("The first valve is opened", "opened"); // Результат: true
```

7.2.5. EQ

Лексикографически сравнивает две строки аналогично оператору "равно".

```
bool String.EQ(string s1, string s2)
```

Параметры

Параметр	Тип	Описание
s1	string	Первая строка.
s2	string	Вторая строка.

Возвращаемое значение

Тип - bool.

true - если строки s1 и s2 совпадают, иначе - false.

ПРИМЕЧАНИЕ

Альтернативные способы сравнения строк на равенство:

- Оператор сравнения: s1 == s2.

Полностью аналогичен функции String.EQ.

- Функция String.Equals.

Может сравнивать строки как с учётом регистра, так и без учёта регистра.

7.2.6. Equals

Проверяет две строки на равенство.

```
bool String.Equals(string x, string y)
bool String.Equals(string x, string y, bool caseSensitive)
```

Параметры

Параметр	Тип	Описание
x	string	Первая строка.
y	string	Вторая строка.
caseSensitive	bool	Опциональный параметр. Чувствительность к регистру символов: - true - символы сравниваются с учётом регистра. - false - строки сравниваются без учёта регистра. Если параметр не указан, строки сравниваются с учётом регистра.

Возвращаемое значение

Тип - bool.

true, если строки x и y совпадают; false - если не совпадают.

ПРИМЕР

```
a: bool = String.Equals("Opened", "opened"); // Результат: false
b: bool = String.Equals("Opened", "opened", false); // Результат: true
```

7.2.7. GE

Лексикографически сравнивает две строки аналогично оператору "больше или равно".

```
bool String.GE(string s1, string s2)
```

Параметры

Параметр	Тип	Описание
s1	string	Первая строка.
s2	string	Вторая строка.

Возвращаемое значение

Тип - bool.

true - если строка s1 больше или равна s2, иначе - false.

ПРИМЕЧАНИЕ
Функция аналогична оператору сравнения строк: s1 >= s2.

7.2.8. GT

Лексикографически сравнивает две строки аналогично оператору "больше".

```
bool String.GT(string s1, string s2)
```

Параметры

Параметр	Тип	Описание
s1	string	Первая строка.
s2	string	Вторая строка.

Возвращаемое значение

Тип - bool.

true - если строка s1 больше s2, иначе - false.

ПРИМЕЧАНИЕ
Функция аналогична оператору сравнения строк: s1 > s2.

7.2.9. IndexOf

Возвращает индекс первого вхождения подстроки в указанную строку.

```
int4 String.IndexOf(string x, string y)
int4 String.IndexOf(string x, string y, int4 start)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, в которой будем искать подстроку.
y	string	Подстрока, которую ищем.
start	int4	Опциональный параметр. Стартовая позиция, с которой начинается поиск. Нумерация начинается с нуля. Если указано значение меньше нуля, принимается равным нулю.

Возвращаемое значение

Тип - int4.

Номер символа в строке x, с которого начинается первое вхождение подстроки y в строку x. Нумерация символов начинается с нуля.

Поиск начинается с позиции start или с начала строки, если start не указано.

Если подстрока не найдена, вернётся -1.

ПРИМЕР

```
a: int4 = String.IndexOf("Warning! Check valve state", "Check");
// Результат: 9
b: int4 = String.IndexOf("one and one more", "one");
// Результат: 0
c: int4 = String.IndexOf("one and one more", "one", 1);
// Результат: 8
d: int4 = String.IndexOf("one and one more", "one", 10);
// Результат: -1
e: int4 = String.IndexOf("Warning!", "warning");
// Результат: -1
```

7.2.10. Insert

Возвращает строку, в указанное место которой вставлена другая строка.

```
string String.Insert(string x, int4 start, string y)
```

Параметры

Параметр	Тип	Описание
x	string	Исходная строка.

Параметр	Тип	Описание
start	int4	Позиция, в которую надо вставить строку. Если значение меньше нуля - считается равной нулю. Если значение больше длины строки x - считается равной длине строки x.
y	string	Строка, которую нужно вставить.

Возвращаемое значение

Тип - string.

Строка x, в которую начиная с позиции start вставлена строка y.

ПРИМЕР

```
a: string = String.Insert("default string", 0, "my ");
// Результат: "my default string"
b: string = String.Insert("default string", 7, " modified");
// Результат: "default modified string"
c: string = String.Insert ("default string", 100, " is short");
// Результат: "default string is short"
```

7.2.11. IsValidFormat

Проверяет, является ли указанная строка корректной строкой формата printf. Описание формата printf: https://en.wikipedia.org/wiki/Printf_format_string

```
bool String.IsValidFormat(string x)
```

Параметры

Параметр	Тип	Описание
x	string	Строка формата.

Возвращаемое значение

Тип - bool.

true, если строка x является корректной строкой в формате printf, иначе - false.

7.2.12. LastIndexOf

Возвращает индекс последнего вхождения подстроки в указанную строку.

```
int4 String.LastIndexOf(string x, string y)
int4 String.LastIndexOf(string x, string y, int4 start)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, в которой будем искать подстроку.
y	string	Подстрока, которую ищем.
start	int4	Опциональный параметр. Стартовая позиция, с которой начинается поиск. Нумерация начинается с нуля. Если указано значение меньше нуля, принимается равным нулю.

Возвращаемое значение

Тип - int4.

Номер символа в строке x, с которого начинается последнее вхождение подстроки y в строку x. Нумерация символов начинается с нуля.

Поиск начинается с позиции start или с конца строки, если start не указано, и идёт в направлении начала строки.

Если start указано, искаться будет такое вхождение, которое заканчивается не позднее позиции start. Иными словами, поиск будет выполняться так, будто после позиции start символов нет.

Если подстрока не найдена, вернётся -1.

ПРИМЕР

```
a: int4 = String.LastIndexOf("Warning! Check valve state", "Check");
// Результат: 9
b: int4 = String.LastIndexOf("one and one more", "one");
// Результат: 8
c: int4 = String.LastIndexOf("one and one more", "one", 2);
// Результат: 0
d: int4 = String.LastIndexOf("one and one more", "one", 1);
// Результат: -1, т.к. подстрока заканчивается после позиции start
e: int4 = String.LastIndexOf("Warning!", "warning");
// Результат: -1
```

7.2.13. LE

Лексикографически сравнивает две строки аналогично оператору "меньше или равно".

```
bool String.LE(string s1, string s2)
```

Параметры

Параметр	Тип	Описание
s1	string	Первая строка.
s2	string	Вторая строка.

Возвращаемое значение

Тип - bool.

true - если строка s1 меньше или равна s2, иначе - false.

ПРИМЕЧАНИЕ

Функция аналогична оператору сравнения строк: s1 <= s2.

7.2.14. Length

Возвращает количество символов в строке.

```
int4 String.Length(string x)
```

Параметры

Параметр	Тип	Описание
x	string	Строка.

Возвращаемое значение

Тип - int4.

Количество символов в строке x.

ПРИМЕР

```
a: int4 = String.Length("Warning!"); // Результат: 8
b: int4 = String.Length(""); // Результат: 0
```

7.2.15. LT

Лексикографически сравнивает две строки аналогично оператору "меньше".

```
bool String.LT(string s1, string s2)
```

Параметры

Параметр	Тип	Описание
s1	string	Первая строка.
s2	string	Вторая строка.

Возвращаемое значение

Тип - bool.

true - если строка s1 меньше s2, иначе - false.

ПРИМЕЧАНИЕ

Функция аналогична оператору сравнения строк: s1 < s2.

7.2.16. NE

Лексикографически сравнивает две строки аналогично оператору "не равно".

```
bool String.NE(string s1, string s2)
```

Параметры

Параметр	Тип	Описание
s1	string	Первая строка.
s2	string	Вторая строка.

Возвращаемое значение

Тип - bool.

true - если строки s1 и s2 различаются; если равны - вернётся false.

ПРИМЕЧАНИЕ

Функция аналогична оператору сравнения строк: s1 != s2.

7.2.17. Remove

Возвращает строку, в которой начиная с указанной позиции удалено указанное количество символов.

```
string String.Remove(string x, int4 start, int4 length)
```

Параметры

Параметр	Тип	Описание
x	string	Исходная строка.
start	int4	Позиция, начиная с которой будем удалять символы. Если значение меньше нуля - считается равной нулю. Если значение больше длины строки - символы не будут удаляться.
length	int4	Количество удаляемых символов. Если значение меньше или равно нулю - символы не будут удаляться. Может быть больше количества символов в строке после позиции start - в этом случае будут удалены все символы до конца строки.

Возвращаемое значение

Тип - string.

Строка x, из которой удалено length символов, начиная с позиции start.

ПРИМЕР

```
a: string = String.Remove("my default string", 3, 8); // Результат: "my string"
b: string = String.Remove("Note: message", -100, 6); // Результат: "message"
c: string = String.Remove("Note: message", 4, 100); // Результат: "Note"
```

7.2.18. Replace

Возвращает строку, в которой все вхождения одной подстроки заменены на другую подстроку.

```
string String.Replace(string x, string oldStr, string newStr)
```

Параметры

Параметр	Тип	Описание
x	string	Исходная строка.

Параметр	Тип	Описание
oldStr	string	Подстрока, которую надо заменить. Если значение - пустая строка, то вернётся исходная строка.
newStr	string	Подстрока, которую надо подставить вместо заменяемой.

Возвращаемое значение

Тип - string.

Строка x, в которой каждое вхождение подстроки oldStr заменено на newStr.

ПРИМЕР

```
a: string = String.Replace("Let's start!", "start", "continue");
// Результат: "Let's continue!"
b: string = String.Replace("Switch 1 turns off device 1.", "1", "2");
// Результат: "Switch 2 turns off device 2."
c: string = String.Replace("Note: Message", "Note: ", "");
// Результат: "Message"
```

7.2.19. Reserve

Возвращает пустую строку с местом, зарезервированным для хранения указанного количества символов. Используется, если внешней функции нужно на вход подать строку определённого размера для записи выходного значения.

```
string String.Reserve(int4 capacity)
```

Параметры

Параметр	Тип	Описание
capacity	int4	Количество символов. Должно быть неотрицательным.

Возвращаемое значение

Тип - string.

Пустая строка с местом, зарезервированным под хранение capacity символов.

ПРИМЕР

```
a: string = String.Reserve(10);
// Результат: "" (пустая строка)
b: int4 = String.Length(a);
// Результат: 10 - количество символов, под которые зарезервировано место.
```

7.2.20. StartsWith

Проверяет, начинается ли строка с указанной подстроки.

```
bool String.StartsWith(string x, string y)
```

Праметры

Параметр	Тип	Описание
x	string	Строка, которую будем проверять.
y	string	Подстрока, которую будем искать в начале строки.

Возвращаемое значение

Тип - bool.

true, если строка x начинается с подстроки y; иначе - false. Если y - пустая строка, вернётся true.

ПРИМЕР

```
a: bool = String.StartsWith("Warning! Pump is broken", "Warning!"); // Результат:
true
```

7.2.21. SubString

Для указанной строки возвращает подстроку.

```
string String.SubString(string x, int4 start)
string String.SubString(string x, int4 start, int4 length)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, в которой будет искаться подстрока.

Параметр	Тип	Описание
start	int4	Номер символа, с которого извлекать подстроку. Нумерация начинается с нуля. Если указано значение меньше нуля, принимается равным нулю.
length	int4	Опциональный параметр. Длина подстроки. Если указан 0 - вернётся пустая подстрока. Если указано значение меньше нуля, принимается равным нулю.

Возвращаемое значение

Тип - string.

Подстрока строки x, начиная с позиции start длины length (если length указан) или до конца строки (если length не указан).

ПРИМЕР

```
a: string = String.SubString("Hello, World!", 0, 5); // Результат: "Hello"
b: string = String.SubString("Hello, World!", 5); // Результат: ", World!"
```

7.2.22. ToBool

Конвертирует указанную строку в значение типа bool.

```
float String.ToBool(string x)
float String.ToBool(string x, bool defValue)
float String.ToBool(string x, bool defValue, bool caseSensitive)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем конвертировать.
defValue	bool	Опциональный параметр. Значение, которое вернётся, если конвертация будет неуспешна. Если значение не указано, считается равным false.
caseSensitive	bool	Опциональный параметр. Чувствительность к регистру. Если не указано, считается равным true.

Возвращаемое значение

Тип - bool.

- true - если x - строка **«true»**.

Если caseSensitive = false, то регистр символов значения не имеет.

- false - если x - строка **«false»**.

Если caseSensitive = false, то регистр символов значения не имеет.

- defValue - если конвертировать не получилось.

ПРИМЕР

```
a: bool = String.ToBool("true"); // Результат: true
b: bool = String.ToBool("True"); // Результат: false
c: bool = String.ToBool("false"); // Результат: false
d: bool = String.ToBool("FALSE", true); // Результат: true
e: bool = String.ToBool("FALSE", true, false); // Результат: false
f: bool = String.ToBool("1"); // Результат: false
g: bool = String.ToBool("non bool"); // Результат: false
h: bool = String.ToBool("non bool"); // Результат: false
i: bool = String.ToBool("non bool", true); // Результат: true
```

7.2.23. ToDouble

Конвертирует указанную строку в число типа double.

```
double String.ToDouble(string x, double defValue)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем конвертировать.
defValue	double	Значение, которое вернётся, если конвертация будет неуспешна.

Возвращаемое значение

Тип - double.

Число, полученное из строки x, или defValue, если конвертировать строку в число нельзя.

ПРИМЕР

```
a: double = String.ToDouble("100.22", 0); // Результат: 100.22
b: double = String.ToDouble("314e-2", 0); // Результат: 3.14
c: double = String.ToDouble("not number", -1); // Результат: -1
d: double = String.ToDouble("1,2", -1); // Результат: -1
```

7.2.24. ToFloat

Конвертирует указанную строку в число типа float.

```
float String.ToFloat(string x, float defValue)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем конвертировать.
defValue	float	Значение, которое вернётся, если конвертация будет неуспешна.

Возвращаемое значение

Тип - float.

Число, полученное из строки x, или defValue, если конвертировать строку в число нельзя.

ПРИМЕР

```
a: float = String.ToFloat("100.22", 0); // Результат: 100.22
b: float = String.ToFloat("314e-2", 0); // Результат: 3.14
c: float = String.ToFloat("not number", -1); // Результат: -1
d: float = String.ToFloat("1,2", -1); // Результат: -1
```

7.2.25. ToInt1

Конвертирует указанную строку в число типа int1.

```
int1 String.ToInt1(string x, int1 defValue)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем конвертировать.
defValue	int1	Значение, которое вернётся, если конвертация будет неуспешна.

Возвращаемое значение

Тип - int1.

Число, полученное из строки x, или defValue, если конвертировать строку в число нельзя.

ПРИМЕР

```
a: int1 = String.ToInt1("100", 0); // Результат: 100
b: int1 = String.ToInt1("not number", -1); // Результат: -1
```

7.2.26. ToInt2

Конвертирует указанную строку в число типа int2.

```
int2 String.ToInt2(string x, int2 defValue)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем конвертировать.
defValue	int2	Значение, которое вернётся, если конвертация будет неуспешна.

Возвращаемое значение

Тип - int2.

Число, полученное из строки x, или defValue, если конвертировать строку в число нельзя.

ПРИМЕР

```
a: int2 = String.ToInt2("100", 0); // Результат: 100
b: int2 = String.ToInt2("not number", -1); // Результат: -1
```

7.2.27. ToInt4

Конвертирует указанную строку в число типа int4.

```
int4 String.ToInt4(string x, int4 defValue)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем конвертировать.
defValue	int4	Значение, которое вернётся, если конвертация будет неуспешна.

Возвращаемое значение

Тип - int4.

Число, полученное из строки x, или defValue, если конвертировать строку в число нельзя.

ПРИМЕР

```
a: int4 = String.ToInt4("100", 0); // Результат: 100
b: int4 = String.ToInt4("not number", -1); // Результат: -1
```

7.2.28. ToInt8

Конвертирует указанную строку в число типа int8.

```
int8 String.ToInt8(string x, int8 defValue)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем конвертировать.
defValue	int8	Значение, которое вернётся, если конвертация будет неуспешна.

Возвращаемое значение

Тип - int8.

Число, полученное из строки x, или defValue, если конвертировать строку в число нельзя.

ПРИМЕР

```
a: int8 = String.ToInt8("100", 0); // Результат: 100
b: int8 = String.ToInt8("not number", -1); // Результат: -1
```

7.2.29. ToLocalizedString

Возвращает строковое представление указанного значения с учётом системного языка. Например, в русском языке у дробных чисел в качестве разделителя будет запятая вместо точки.

```
string String.ToLocalizedString(bool x)
string String.ToLocalizedString(int1 x)
string String.ToLocalizedString(string format, int1 x)
string String.ToLocalizedString(int2 x)
```

```
string String.ToLocalizedString(string format, int2 x)
string String.ToLocalizedString(int4 x)
string String.ToLocalizedString(string format, int4 x)
string String.ToLocalizedString(int8 x)
string String.ToLocalizedString(string format, int8 x)
string String.ToLocalizedString(uint1 x)
string String.ToLocalizedString(string format, uint1 x)
string String.ToLocalizedString(uint2 x)
string String.ToLocalizedString(string format, uint2 x)
string String.ToLocalizedString(uint4 x)
string String.ToLocalizedString(string format, uint4 x)
string String.ToLocalizedString(uint8 x)
string String.ToLocalizedString(string format, uint8 x)
string String.ToLocalizedString(float x)
string String.ToLocalizedString(string format, float x)
string String.ToLocalizedString(double x)
string String.ToLocalizedString(string format, double x)
```

Параметры

Параметр	Тип	Описание
format	string	Опциональный параметр. Правило преобразования чисел в строку в формате printf: https://en.wikipedia.org/wiki/Printf_format_string Если не указан, используется значение %f.
x	bool, int1, int2, int4, int8, uint1, uint2, uint4, uint8, float, double	Значение, которое нужно преобразовать к строке.

ОБРАТИТЕ ВНИМАНИЕ

Формат вывода %f, который используется по умолчанию, имеет точность 6 знаков после запятой. Если нужно преобразовывать числа в строку с большей точностью, укажите формат вывода с большей точностью. Например, %.10f - выводить дробные числа с точностью 10 знаков после запятой.

Возвращаемое значение

Тип - string.

Строковое представление значения x:

- если x имеет тип bool - возвращается строка «**true**» или «**false**».
- если x - число, возвращается строковое представление этого числа в формате format с учётом системного языка.

ПРИМЕР

```
a: string = String.ToLocalizedString(7.15);
// Результат:
// - "7.150000", если системный язык - английский
// - "7,150000", если системный язык - русский
```

7.2.30. ToLower

Возвращает строку, в которой все символы приведены к нижнему регистру.

```
string String.ToLower(string x)
```

Параметры

Параметр	Тип	Описание
x	string	Исходная строка.

Возвращаемое значение

Тип - string.

Строка x, в которой все символы приведены к нижнему регистру.

ПРИМЕР

```
a: string = String.ToLower("ALARM!"); // Результат: "alarm!"
```

7.2.31. ToString

Возвращает строковое представление указанного значения.

```
string String.ToString(bool x)
string String.ToString(int8 x)
string String.ToString(string format, int8 x)
string String.ToString(uint8 x)
string String.ToString(string format, uint8 x)
string String.ToString(double x)
string String.ToString(string format, double x)
```

Параметры

Параметр	Тип	Описание
format	string	Опциональный параметр. Правило преобразования чисел в строку в формате printf: https://en.wikipedia.org/wiki/Printf_format_string Если не указан, используется значение %f.
x	int8, uint8, double, bool	Значение, которое нужно преобразовать к строке.

ОБРАТИТЕ ВНИМАНИЕ

Формат вывода %f, который используется по умолчанию, имеет точность 6 знаков после запятой. Если нужно преобразовывать числа в строку с большей точностью, укажите формат вывода с большей точностью. Например, %.10f - выводить дробные числа с точностью 10 знаков после запятой.

Возвращаемое значение

Тип - string.

Строковое представление значения x:

- если x имеет тип bool - возвращается строка **«true»** или **«false»**.
- если x - число, возвращается строковое представление этого числа в формате format.

ПРИМЕР

Преобразование целых чисел:

```
a: string = String.ToString(777); // Результат: "777"
b: string = String.ToString("%05f", 12); // Результат: "00012"
c: string = String.ToString("%05f", -12); // Результат: "-0012"
```

ПРИМЕР

Преобразование вещественных чисел:

```
a: string = String.ToString(1.1); // Результат: "1.100000"
b: string = String.ToString(-123.0e-2); // Результат: "-1.230000"
c: string = String.ToString("%e", 0.001);
// Результат: "1.000000e-03" (экспоненциальная форма записи)
d: string = String.ToString(0.123456789);
// Результат: "0.123457" (значение округляется до 6 знаков после запятой)
e: string = String.ToString("%.10f", 0.123456789);
// Результат: "0.1234567890" (значение выводится с точностью 10 знаков после запятой)
```

ПРИМЕР

Преобразование логических значений:

```
a: string = String.ToString(true); // Результат: "true"
```

7.2.32. ToUint1

Конвертирует указанную строку в число типа uint1.

```
uint2 String.ToUint1(string x, uint1 defValue)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем конвертировать.
defValue	uint1	Значение, которое вернётся, если конвертация будет неуспешна.

Возвращаемое значение

Тип - uint1.

Число, полученное из строки x, или defValue, если конвертировать строку в число нельзя.

ПРИМЕР

```
a: uint2 = String.ToUint1("100", 0); // Результат: 100
b: uint2 = String.ToUint1("not number", 0); // Результат: 0
```

7.2.33. ToUint2

Конвертирует указанную строку в число типа uint2.

```
uint2 String.ToUint2(string x, uint2 defValue)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем конвертировать.
defValue	uint2	Значение, которое вернётся, если конвертация будет неуспешна.

Возвращаемое значение

Тип - uint2.

Число, полученное из строки x, или defValue, если конвертировать строку в число нельзя.

ПРИМЕР

```
a: uint2 = String.ToUint2("100", 0); // Результат: 100
b: uint2 = String.ToUint2("not number", 0); // Результат: 0
```

7.2.34. ToUint4

Конвертирует указанную строку в число типа uint4.

```
uint4 String.ToUint4(string x, uint4 defValue)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем конвертировать.
defValue	uint4	Значение, которое вернётся, если конвертация будет неуспешна.

Возвращаемое значение

Тип - uint4.

Число, полученное из строки x, или defValue, если конвертировать строку в число нельзя.

ПРИМЕР

```
a: uint4 = String.ToUint4("100", 0); // Результат: 100
b: uint4 = String.ToUint4("not number", 0); // Результат: 0
```

7.2.35. ToUint8

Конвертирует указанную строку в число типа uint8.

```
uint8 String.ToUint8(string x, uint8 defValue)
```

Параметры

Параметр	Тип	Описание
x	string	Строка, которую будем конвертировать.
defValue	uint8	Значение, которое вернётся, если конвертация будет неуспешна.

Возвращаемое значение

Тип - uint8.

Число, полученное из строки x, или defValue, если конвертировать строку в число нельзя.

ПРИМЕР

```
a: uint8 = String.ToUint8("100", 0); // Результат: 100
b: uint8 = String.ToUint8("not number", 0); // Результат: 0
```

7.2.36. ToUpper

Возвращает строку, в которой все символы приведены к верхнему регистру.

```
string String.ToUpper(string x)
```

Параметры

Параметр	Тип	Описание
x	string	Исходная строка.

Возвращаемое значение

Тип - string.

Строка x, в которой все символы приведены к верхнему регистру.

ПРИМЕР

```
a: string = String.ToLower("alarm!"); // Результат: "ALARM!"
```

7.2.37. Trim

Возвращает строку, в которой удалены пробелы в начале и в конце строки.

```
string String.Trim(string x)
```

Параметры

Параметр	Тип	Описание
x	string	Исходная строка.

Возвращаемое значение

Тип - string.

Строка x, в которой удалены все пробелы в начале и в конце строки.

ПРИМЕР

```
a: string = String.Trim("  Warning!  "); // Результат: "Warning!"
```

7.3. Функции обработки времени

Функций обработки времени вызываются обращением к пространству имен DateTime.

Функции обработки времени получают в качестве аргумента или возвращают метку времени - значение типа uint8, представляющее собой количество 100-наносекундных интервалов прошедших с 1 января 1601 года.

Функция	Описание
AddDays	Возвращает метку времени, смещённую на указанное количество дней.
AddHours	Возвращает метку времени, смещённую на указанное количество часов.
AddMinutes	Возвращает метку времени, смещённую на указанное количество минут.
AddMonths	Возвращает метку времени, смещённую на указанное количество месяцев.
AddMSeconds	Возвращает метку времени, смещённую на указанное количество миллисекунд.
AddSeconds	Возвращает метку времени, смещённую на указанное количество секунд.
AddYears	Возвращает метку времени, смещённую на указанное количество лет.
Create	Возвращает метку времени, полученную из набора указанных значений.
Day	Для указанной метки времени возвращает число месяца.
DayOfWeek	Для указанной метки времени возвращает день недели.
DaysInMonth	Возвращает количество дней в указанном месяце указанного года.

Функция	Описание
Hour	Для указанной метки времени возвращает час.
IsLeapYear	Проверяет, является ли указанный год високосным.
Minute	Для указанной метки времени возвращает минуты.
Month	Для указанной метки времени возвращает месяц.
MSecond	Для указанной метки времени возвращает миллисекунды.
Now	Возвращает текущее время компьютера.
Parse	Возвращает метку времени, полученную из указанной строки.
Second	Для указанной метки времени возвращает секунды.
ToLocal	Для указанной метки времени со временем по UTC возвращает метку времени с соответствующим временем в текущем часовом поясе.
ToString	Для указанной метки времени возвращает дату-время в виде строки.
ToUtc	Для указанной метки времени в текущем часовом поясе возвращает метку времени с соответствующим временем по UTC.
UtcNow	Возвращает текущее время по UTC.
Year	Для указанной метки времени возвращает год.

7.3.1. AddDays

Возвращает метку времени, смещённую на указанное количество дней.

```
uint8 DateTime.AddDays(uint8 timestamp, double days)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени, от которой смещаться.
days	double	Количество дней, на которое надо сместиться. Значение может быть отрицательным (смещение назад во времени). Можно указать дробное смещение.

Возвращаемое значение

Тип - uint8.

Метка времени, смещённая относительно timestamp на указанное количество дней.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:00:00");
a: uint8 = DateTime.AddDays(timestamp, 2);
// Результат: метка времени, соответствующая "08.09.1993 12:00:00"
b: uint8 = DateTime.AddDays(timestamp, -15);
// Результат: метка времени, соответствующая "22.08.1993 12:00:00"
c: uint8 = DateTime.AddDays(timestamp, 0.25);
// Результат: метка времени, соответствующая "06.09.1993 18:00:00"
```

7.3.2. AddHours

Возвращает метку времени, смещённую на указанное количество часов.

```
uint8 DateTime.AddHours(uint8 timestamp, double hours)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени, от которой смещаться.
hours	double	Количество часов, на которое надо сместиться. Значение может быть отрицательным (смещение назад во времени). Можно указать дробное смещение.

Возвращаемое значение

Тип - uint8.

Метка времени, смещённая относительно timestamp на указанное количество часов.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:00:00");
a: uint8 = DateTime.AddHours(timestamp, 2);
// Результат: метка времени, соответствующая "06.09.1993 14:00:00"
b: uint8 = DateTime.AddHours(timestamp, -15);
// Результат: метка времени, соответствующая "05.09.1993 21:00:00"
c: uint8 = DateTime.AddHours(timestamp, 1.5);
// Результат: метка времени, соответствующая "06.09.1993 13:30:00"
```

7.3.3. AddMinutes

Возвращает метку времени, смещённую на указанное количество минут.

```
uint8 DateTime.AddMinutes(uint8 timestamp, double minutes)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени, от которой смещаться.
minutes	double	Количество минут, на которое надо сместиться. Значение может быть отрицательным (смещение назад во времени). Можно указать дробное смещение.

Возвращаемое значение

Тип - uint8.

Метка времени, смещённая относительно timestamp на указанное количество минут.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:00:00");
a: uint8 = DateTime.AddMinutes(timestamp, 30);
// Результат: метка времени, соответствующая "06.09.1993 12:30:00"
b: uint8 = DateTime.AddMinutes(timestamp, -90);
// Результат: метка времени, соответствующая "06.09.1993 10:30:00"
c: uint8 = DateTime.AddMinutes(timestamp, 1.5);
// Результат: метка времени, соответствующая "06.09.1993 12:01:30"
```

7.3.4. AddMonths

Возвращает метку времени, смещённую на указанное количество месяцев.

```
uint8 DateTime.AddMonths(uint8 timestamp, int4 months)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени, от которой смещаться.
months	int4	Количество месяцев, на которое надо сместиться. Значение может быть отрицательным (смещение назад во времени).

Возвращаемое значение

Тип - uint8.

Метка времени, смещённая относительно timestamp на указанное количество месяцев.
Если при смещении получилась недопустимая дата (например, 30 февраля), то в качестве даты будет взят последний день полученного месяца; остальные поля метки времени (часы, минуты и т.д.) останутся, как у исходной метки времени.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:00:00");
a: uint8 = DateTime.AddMonths(timestamp, 2);
// Результат: метка времени, соответствующая "06.11.1993 12:00:00"
b: uint8 = DateTime.AddMonths(timestamp, 6);
// Результат: метка времени, соответствующая "06.03.1994 12:00:00"
c: uint8 = DateTime.AddMonths(timestamp, -3);
// Результат: метка времени, соответствующая "06.06.1993 12:00:00"
```

ПРИМЕР

Если при смещении получили недопустимую дату:

```
in_jan: uint8 = DateTime.Parse("31.01.2000 12:30:00");
in_feb: uint8 = DateTime.AddMonths(in_jan, 1);
// Результат: метка времени, соответствующая "29.02.2000 12:30:00"
```

7.3.5. AddMSeconds

Возвращает метку времени, смещённую на указанное количество миллисекунд.

```
uint8 DateTime.AddMSeconds(uint8 timestamp, double mseconds)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени, от которой смещаться.
mseconds	double	Количество миллисекунд, на которое надо сместиться. Значение может быть отрицательным (смещение назад во времени). Можно указать дробное смещение.

Возвращаемое значение

Тип - uint8.

Метка времени, смещённая относительно timestamp на указанное количество миллисекунд.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:00:00");
a: uint8 = DateTime.AddMSeconds(timestamp, 500);
// Результат: метка времени, соответствующая "06.09.1993 12:00:00.500"
b: uint8 = DateTime.AddMSeconds(timestamp, -2500);
// Результат: метка времени, соответствующая "06.09.1993 11:59:57.500"
```

7.3.6. AddSeconds

Возвращает метку времени, смещённую на указанное количество секунд.

```
uint8 DateTime.AddSeconds(uint8 timestamp, double seconds)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени, от которой смещаться.
seconds	double	Количество секунд, на которое надо сместиться. Значение может быть отрицательным (смещение назад во времени). Можно указать дробное смещение.

Возвращаемое значение

Тип - uint8.

Метка времени, смещённая относительно timestamp на указанное количество секунд.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:00:00");
a: uint8 = DateTime.AddSeconds(timestamp, 30);
// Результат: метка времени, соответствующая "06.09.1993 12:00:30"
b: uint8 = DateTime.AddSeconds(timestamp, -90);
// Результат: метка времени, соответствующая "06.09.1993 11:58:30"
c: uint8 = DateTime.AddSeconds(timestamp, 50.5);
// Результат: метка времени, соответствующая "06.09.1993 12:00:50.500"
```

7.3.7. AddYears

Возвращает метку времени, смещённую на указанное количество лет.

```
uint8 DateTime.AddYears(uint8 timestamp, int4 years)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени, от которой смещаться.
years	int4	Количество лет, на которое надо сместиться. Значение может быть отрицательным (смещение назад во времени).

Возвращаемое значение

Тип - uint8.

Метка времени, смещённая относительно timestamp на указанное количество лет.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:00:00");
a: uint8 = DateTime.AddYears(timestamp, 7);
// Результат: метка времени, соответствующая "06.09.2000 12:00:00"
b: uint8 = DateTime.AddYears(timestamp, -13);
// Результат: метка времени, соответствующая "06.09.1980 12:00:00"
```

7.3.8. Create

Возвращает метку времени, полученную из набора указанных значений.

```
uint8 DateTime.Create(uint2 year, uint1 month, uint1 day, uint1 h, uint1 m, uint1 s)
uint8 DateTime.Create(uint2 year, uint1 month, uint1 day, uint1 h, uint1 m, uint1 s, uint2
ms)
```

Параметры

Параметр	Тип	Описание
year	uint2	Год. 1601 ≤ year
month	uint1	Месяц. 1 ≤ month ≤ 12
day	uint1	Число месяца. 1 ≤ day ≤ кол-во дней в месяце
h	uint1	Час. 0 ≤ h ≤ 23

Параметр	Тип	Описание
m	uint1	Минуты. 0 ≤ m ≤ 59
s	uint1	Секунды. 0 ≤ s ≤ 59
ms	uint2	Миллисекунды. Опциональный параметр: если не указан, считается равным 0.

Возвращаемое значение

Тип - uint8.

Метка времени, составленная из набора указанных значений года, месяца, даты, часа, минут, секунд и миллисекунд.

Если какое-то значение принимает некорректное значение, возвращается нулевая метка времени - 1 января 1601 года, 0:00:00.000.

ПРИМЕР

```
a: uint8 = DateTime.Create(2000, 1, 2, 3, 4, 5);
// Результат: метка времени 2 января 2000 года, 3:04:05
b: uint8 = DateTime.Create(1993, 9, 6, 12, 5, 15, 301);
// Результат: метка времени 6 сентября 1993 года, 12:05:15.301
c: uint8 = DateTime.Create(1999, 2, 29, 12, 0, 0, 0);
// Результат: 1 января 1601 года, 0:00:00.000, т.к. в феврале 1999 года 28 дней.
```

7.3.9. Day

Для указанной метки времени возвращает число месяца.

```
uint1 DateTime.Day(uint8 timestamp)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени.

Возвращаемое значение

Тип - uint1.

Число месяца метки времени timestamp. 1 ≤ значение ≤ 31.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:00:00");
Result: uint1 = DateTime.Day(timestamp);
// Результат: 6
```

7.3.10. DayOfWeek

Для указанной метки времени возвращает день недели.

Для указанной метки времени возвращает день недели (0-6). 0 - воскресенье, 1 - понедельник и т.д.

```
uint1 DateTime.DayOfWeek(uint8 timestamp)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени.

Возвращаемое значение

Тип - uint1.

День недели метки времени timestamp. 0 ≤ значение ≤ 6: 0 - воскресенье, 1 - понедельник и т.д.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:25:31");
Result: uint1 = DateTime.DayOfWeek(timestamp);
// Результат: 1 (понедельник)
```

7.3.11. DaysInMonth

Возвращает количество дней в указанном месяце указанного года.

```
uint1 DateTime.DaysInMonth(uint2 year, uint1 month)
```

Параметры

Параметр	Тип	Описание
year	uint2	Год.

Параметр	Тип	Описание
month	uint1	Месяц.

Возвращаемое значение

Тип - uint1.

Количество дней в указанном месяце указанного года:

- для февраля - количество дней зависит от того, является ли год високосным.
- для остальных месяцев - количество дней не зависит от года.

7.3.12. Hour

Для указанной метки времени возвращает час.

```
uint1 DateTime.Hour(uint8 timestamp)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени.

Возвращаемое значение

Тип - uint1.

Час метки времени timestamp. 0 ≤ значение ≤ 23.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:00:00");
Result: uint1 = DateTime.Hour(timestamp);
// Результат: 12
```

7.3.13. IsLeapYear

Проверяет, является ли указанный год високосным.

```
bool DateTime.IsLeapYear(uint2 year)
```

Параметры

Параметр	Тип	Описание
year	uint2	Год.

Возвращаемое значение

Тип - bool.

true - если год високосный, иначе - false.

7.3.14. Minute

Для указанной метки времени возвращает минуты.

```
uint1 DateTime.Minute(uint8 timestamp)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени.

Возвращаемое значение

Тип - uint1.

Минуты метки времени timestamp. 0 ≤ значение ≤ 59.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:25:00");
Result: uint1 = DateTime.Minute(timestamp);
// Результат: 25
```

7.3.15. Month

Для указанной метки времени возвращает месяц.

```
uint1 DateTime.Month(uint8 timestamp)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени.

Возвращаемое значение

Тип - uint1.

Месяц метки времени timestamp. 1 ≤ значение ≤ 12.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:00:00");
Result: uint1 = DateTime.Month(timestamp);
// Результат: 9
```

7.3.16. MSecond

Для указанной метки времени возвращает миллисекунды.

```
uint2 DateTime.MSecond(uint8 timestamp)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени.

Возвращаемое значение

Тип - uint2.

Миллисекунды метки времени timestamp. 0 ≤ значение ≤ 999.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:25:31.155");
Result: uint2 = DateTime.MSecond(timestamp);
// Результат: 155
```

7.3.17. Now

Возвращает текущее время компьютера.

```
uint8 DateTime.Now()
```

Возвращаемое значение

Тип - uint8.

Метка времени с текущим временем.

7.3.18. Parse

Возвращает метку времени, полученную из указанной строки.

```
uint8 DateTime.Parse(string date_time)
uint8 DateTime.Parse(string date_time, string format)
```

Параметры

Параметр	Тип	Описание
date_time	string	Строка, описывающая время.
format	string	Опциональный параметр. Формат даты/времени, согласно которому будет разбираться строка (стр. 134).

Возвращаемое значение

Тип - uint8.

Метка времени, соответствующая описанным в строке дате/времени. Если в строке указана только дата, возвращает полночь указанной даты: 00:00:00. Если строку не удаётся разобрать, возвращает нулевую метку времени: 1 января 1601 года, 0:00:00.000.

ПРИМЕР

```
a: uint8 = DateTime.Parse("06.09.1993 12:01:01");
// Результат: метка времени, соответствующая указанному в строке моменту времени.
b: uint8 = DateTime.Parse("12:01:01 06.09.1993", "%H:%M:S %d.%m.%Y");
// Результат: метка времени, разобранная согласно указанному формату
```

7.3.19. Second

Для указанной метки времени возвращает секунды.

```
uint1 DateTime.Second(uint8 timestamp)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени.

Возвращаемое значение

Тип - uint1.

Секунды метки времени timestamp. 0 ≤ значение ≤ 59.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:25:31");
Result: uint1 = DateTime.Second(timestamp);
// Результат: 31
```

7.3.20. ToLocal

Для указанной метки времени со временем по UTC возвращает метку времени с соответствующим временем в текущем часовом поясе.

```
uint8 DateTime.ToLocal(uint8 utc_time)
```

Параметры

Параметр	Тип	Описание
utc_time	uint8	Метка времени по UTC.

Возвращаемое значение

Тип - uint8.

Метка времени в текущем часовом поясе, соответствующая времени utc_time по UTC.

7.3.21. ToString

Для указанной метки времени возвращает дату-время в виде строки.

```
string DateTime.ToString(uint8 timestamp)
string DateTime.ToString(uint8 timestamp, string format)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени.
format	string	Опциональный параметр. Формат даты/времени, согласно которому будет формироваться строка (стр. 134).

Возвращаемое значение

Тип - string.

Строка, описывающая дату-время, в указанном формате или, если формат не указан, в формате %d.%m.%Y %H:%M:%S

ПРИМЕР

```
timestamp: uint8 = DateTime.Create(2000, 1, 2, 3, 4, 5, 0);
a: string = DateTime.ToString(timestamp);
// Результат: "02.01.2000 03:04:05"
b: string = DateTime.ToString(timestamp, "%d/%m/%Y-%H:%M:%S.%f");
// Результат: "02/01/2000-03:04:05.000000" (строка в указанном формате)
```

7.3.22. ToUtc

Для указанной метки времени в текущем часовом поясе возвращает метку времени с соответствующим временем по UTC.

```
uint8 DateTime.ToUtc(uint8 local)
```

Параметры

Параметр	Тип	Описание
local	uint8	Метка времени в текущем часовом поясе.

Возвращаемое значение

Тип - uint8.

Метка времени по UTC, соответствующая времени local в текущем часовом поясе.

7.3.23. UtcNow

Возвращает текущее время по UTC.

```
uint8 DateTime.UtcNow()
```

Возвращаемое значение

Тип - uint8.

Метка времени с текущим временем по UTC.

7.3.24. Year

Для указанной метки времени возвращает год.

```
uint2 DateTime.Year(uint8 timestamp)
```

Параметры

Параметр	Тип	Описание
timestamp	uint8	Метка времени.

Возвращаемое значение

Тип - uint2.

Год метки времени timestamp. Значение ≥ 1601.

ПРИМЕР

```
timestamp: uint8 = DateTime.Parse("06.09.1993 12:00:00");
Result: uint2 = DateTime.Year(timestamp);
// Результат: 1993
```

7.4. Функции типа variant

Вызов внутренних функций типа variant осуществляется обращением к пространству имен Variant.

Функция/группа функций	Описание
Add	Возвращает сумму двух вариантов.
And	Возвращает побитовое И двух вариантов.
DEC	Возвращает значение варианта, уменьшенное на 1.

Функция/группа функций	Описание
Div	Возвращает частное двух вариантов.
EQ	Сравнивает два варианта аналогично оператору "равно".
From<T>	Группа функций, которые создают значение типа variant из значения элементарного типа.
GE	Сравнивает два варианта аналогично оператору "больше или равно".
GT	Сравнивает два варианта аналогично оператору "больше".
INC	Возвращает значение варианта, увеличенное на 1.
INV	Возвращает побитовую инверсию варианта.
Is<T>	Группа функций, которые проверяют, относится ли значение типа variant к элементарному типу.
IsEmpty	Проверяет, является ли значение типа variant пустым.
LE	Сравнивает два варианта аналогично оператору "меньше или равно".
LogicalAnd	Возвращает логическое И двух вариантов.
LogicalOr	Возвращает логическое ИЛИ двух вариантов.
LT	Сравнивает два варианта аналогично оператору "меньше".
MayConvertTo<T>	Группа функций, которые проверяют, можно ли конвертировать значение типа variant в элементарный тип.
Mul	Возвращает произведение двух варинатов.
NE	Сравнивает два варианта аналогично оператору "не равно".
NEG	Возвращает значение указанного варианта с противоположным знаком.
NOT	Возвращает логическое отрицание указанного варианта.
Or	Возвращает побитовое ИЛИ двух вариантов.
Sub	Возвращает разность двух вариантов.
To<T>	Группа функций, которые для значения типа variant возвращают её значение, конвертированное в элементарный тип.
Xor	Возвращает побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ двух вариантов.

7.4.1. Add

Возвращает сумму двух вариантов.

```
variant Variant.Add(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первое слагаемое.
y	variant	Второе слагаемое.

Возвращаемое значение

Тип - variant.

Сумма x и y. Если типы значений несовместимы, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

Сумма двух вариантов с целыми значениями:

```
x: variant = 40;
y: variant = 10;
res: variant = Variant.Add(x, y);
// Результат: 50
```

ПРИМЕР

Сумма двух вариантов с несовместимыми значениями:

```
x: variant = 40;
y: variant = "10";
res: variant = Variant.Add(x, y);
// Результат: VT_EMPTY
```

7.4.2. And

Возвращает побитовое И двух вариантов.

```
variant Variant.And(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый вариант. Должен иметь целочисленное значение.
y	variant	Второй вариант. Должен иметь целочисленное значение.

Возвращаемое значение

Тип - variant.

Побитовое И параметров x и y. Если любой из параметров содержит не целочисленное значение, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

Побитовое И двух вариантов с целочисленными значениями:

```
x: variant = 12; // В двоичной записи - 1100
y: variant = 21; // В двоичной записи - 10101
res: variant = Variant.And(x, y);
// Результат: 4 (в двоичной записи - 100)
```

ПРИМЕР

Побитовое И двух вариантов с несовместимыми значениями:

```
x: variant = 12;
y: variant = 1.0;
res: variant = Variant.And(x, y);
// Результат: VT_EMPTY
```

7.4.3. DEC

Возвращает значение варианта, уменьшенное на 1.

```
variant Variant.DEC(variant x)
```

Параметры

Параметр	Тип	Описание
x	variant	Вариант. Должен иметь целочисленное значение.

Возвращаемое значение

Тип - variant.

Значение x, уменьшенное на 1. Если x имеет не целочисленный тип, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

```
x: variant = 15;
res: variant = Variant.DEC(x);
// Результат: 14
```

ПРИМЕР

```
x: variant = 2.5;
res: variant = Variant.DEC(x);
// Результат: VT_EMPTY
```

7.4.4. Div

Возвращает частное двух вариантов.

```
variant Variant.Div(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Делимое.
y	variant	Делитель.

Возвращаемое значение

Тип - variant.

Частное x и y. Если значения целочисленные, вернётся целая часть от деления. Если типы значений несовместимы, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

Частное двух вариантов с целыми значениями:

```
x: variant = 45;
y: variant = 10;
res: variant = Variant.Div(x, y);
// Результат: 4
```

ПРИМЕР

Частное двух вариантов с вещественными значениями:

```
x: variant = 45.0;
y: variant = 10.0;
res: variant = Variant.Div(x, y);
// Результат: 4.5
```

ПРИМЕР

Частное двух вариантов с несовместимыми значениями:

```
x: variant = 40;
y: variant = "10";
res: variant = Variant.Div(x, y);
// Результат: VT_EMPTY
```

7.4.5. EQ

Сравнивает два варианта аналогично оператору "равно".

```
variant Variant.EQ(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый вариант.
y	variant	Второй вариант.

Возвращаемое значение

Тип - variant.

Результат сравнения x == y:

- true - если операция истинна для значений.
- false - если операция ложна для значений.
- VT_EMPTY - если значения несравнимы или один из операндов - пустое значение (VT_EMPTY).

ПРИМЕР

```
x: variant = 40;
y: variant = 40.0;
res: variant = Variant.EQ(x, y);
// Результат: true
```

ПРИМЕР

```
x: variant = 40;
y: variant = "40";
res: variant = Variant.EQ(x, y);
// Результат: VT_EMPTY
```

7.4.6. From<T>

Группа функций, которые создают значение типа variant из значения элементарного типа.

```
variant Variant.FromInt1(int1 x)
variant Variant.FromInt2(int2 x)
variant Variant.FromInt4(int4 x)
variant Variant.FromInt8(int8 x)
variant Variant.FromUint1(uint1 x)
variant Variant.FromUint2(uint2 x)
variant Variant.FromUint4(uint4 x)
variant Variant.FromUint8(uint8 x)
variant Variant.FromFloat(float x)
variant Variant.FromDouble(double x)
variant Variant.FromBool(bool x)
variant Variant.FromString(string x)
```

Параметры

Параметр	Тип	Описание
x	все кроме variant	Значение, которое будет присвоено варианту.

Возвращаемое значение

Тип - variant.

Вариант (значение типа variant), имеющий значение x.

ПРИМЕР

Создать переменную типа variant из переменной элементарного типа uint1:

```
X1: uint1 = 100;
V1: variant = Variant.FromUint1(X1);
```

ПРИМЕР

Создать переменную типа variant из переменной элементарного типа int4:

```
X2: int4 = 12300655;
V2: variant = Variant.FromInt4(X2);
```

ПРИМЕР

Создать переменную типа variant из переменной элементарного типа double:

```
X3: double = 345.543333;
V3: variant = Variant.FromDouble(X3);
```

ПРИМЕР

Создать переменную типа variant из переменной элементарного типа string:

```
X4: string = "my string";
V4: variant = Variant.FromString(X4);
```

ПРИМЕР

Создать переменную типа variant из переменной элементарного типа bool:

```
X5: bool = false;
V5: variant = Variant.FromBool(X5);
```

7.4.7. GE

Сравнивает два варианта аналогично оператору "больше или равно".

```
variant Variant.GE(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый вариант.

Параметр	Тип	Описание
y	variant	Второй вариант.

Возвращаемое значение

Тип - variant.

Результат сравнения x >= y:

- true - если операция истинна для значений.
- false - если операция ложна для значений.
- VT_EMPTY - если значения несравнимы или один из операндов - пустое значение (VT_EMPTY).

ПРИМЕР

```
x: variant = 40;
y: variant = 40.0;
res: variant = Variant.GE(x, y);
// Результат: true
```

ПРИМЕР

```
x: variant = 40;
y: variant = "40";
res: variant = Variant.GE(x, y);
// Результат: VT_EMPTY
```

7.4.8. GT

Сравнивает два варианта аналогично оператору "больше".

```
variant Variant.GT(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый вариант.
y	variant	Второй вариант.

Возвращаемое значение

Тип - variant.

Результат сравнения x > y:

- true - если операция истинна для значений.
- false - если операция ложна для значений.
- VT_EMPTY - если значения несравнимы или один из операндов - пустое значение (VT_EMPTY).

ПРИМЕР

```
x: variant = 40;
y: variant = 40.0;
res: variant = Variant.GT(x, y);
// Результат: false
```

ПРИМЕР

```
x: variant = 40;
y: variant = "40";
res: variant = Variant.GT(x, y);
// Результат: VT_EMPTY
```

7.4.9. INC

Возвращает значение варианта, увеличенное на 1.

```
variant Variant.INC(variant x)
```

Параметры

Параметр	Тип	Описание
x	variant	Вариант. Должен иметь целочисленное значение.

Возвращаемое значение

Тип - variant.

Значение x, увеличенное на 1. Если x имеет не целочисленный тип, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

```
x: variant = 15;
res: variant = Variant.INC(x);
// Результат: 16
```

ПРИМЕР

```
x: variant = 2.5;
res: variant = Variant.INC(x);
// Результат: VT_EMPTY
```

7.4.10. INV

Возвращает побитовую инверсию варианта.

```
variant Variant.INV(variant x)
```

Параметры

Параметр	Тип	Описание
x	variant	Вариант. Должен иметь беззнаковое целое значение.

Возвращаемое значение

Тип - variant.

Побитовая инверсия варианта x. Если x имеет не беззнаковый целый тип, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

Побитовая инверсия беззнакового целого значения:

```
x: variant = Variant.FromUint1(99); // В двоичной записи - 01100011
res: variant = Variant.INV(x);
// Результат: 156 (в двоичной записи - 10011100)
```

ПРИМЕР

Побитовая инверсия не беззнакового целого значения:

```
x: variant = 1.15;
res: variant = Variant.INV(x);
// Результат: VT_EMPTY
```

7.4.11. Is<T>

Группа функций, которые проверяют, относится ли значение типа variant к элементарному типу.

```
bool Variant.IsInt1(variant x)
bool Variant.IsInt2(variant x)
bool Variant.IsInt4(variant x)
bool Variant.IsInt8(variant x)
bool Variant.IsUint1(variant x)
bool Variant.IsUint2(variant x)
bool Variant.IsUint4(variant x)
bool Variant.IsUint8(variant x)
bool Variant.IsFloat(variant x)
bool Variant.IsDouble(variant x)
bool Variant.IsBool(variant x)
bool Variant.IsString(variant x)
```

Параметры

Параметр	Тип	Описание
x	variant	Вариант, который будем проверять.

Возвращаемое значение

Тип - bool.

true - если значение варианта x является значением элементарного типа; false - если не является.

ПРИМЕР

Проверить соответствует ли переменная типу uint1:

```
X1: variant = 25;
Result1: bool = Variant.IsUint1(X1);
// Результат: true
```

ПРИМЕР

Проверить, соответствует ли переменная типу float:

```
X2: variant = "my string";
Result2: bool = Variant.IsFloat(X2);
// Результат: false
```

7.4.12. IsEmpty

Проверяет, является ли значение типа variant пустым.

```
bool Variant.IsEmpty(variant x)
```

Параметры

Параметр	Тип	Описание
x	variant	Вариант, который будем проверять.

Возвращаемое значение

Тип - bool.

true - если значение варианта x пустое (VT_EMPTY); false - если не пустое.

ПРИМЕР

```
x: variant; // переменная типа variant без инициализации (VT_EMPTY)
Result2: bool = Variant.IsEmpty(x);
// Результат: true
```

7.4.13. LE

Сравнивает два варианта аналогично оператору "меньше или равно".

```
variant Variant.LE(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый вариант.
y	variant	Второй вариант.

Возвращаемое значение

Тип - variant.

Результат сравнения x <= y:

- true - если операция истинна для значений.
- false - если операция ложна для значений.
- VT_EMPTY - если значения несравнимы или один из операндов - пустое значение (VT_EMPTY).

ПРИМЕР

```
x: variant = 40;
y: variant = 40.0;
res: variant = Variant.LE(x, y);
// Результат: true
```

ПРИМЕР

```
x: variant = 40;
y: variant = "40";
res: variant = Variant.LE(x, y);
// Результат: VT_EMPTY
```

7.4.14. LogicalAnd

Возвращает логическое И двух вариантов.

```
variant Variant.LogicalAnd(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый вариант. Должен иметь значение типа bool.
y	variant	Второй вариант. Должен иметь значение типа bool.

Возвращаемое значение

Тип - variant.

Логическое И параметров x и y. Если любой из параметров содержит значение не типа bool, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

Логическое И двух вариантов с логическими значениями:

```
x: variant = true;
y: variant = false;
res: variant = Variant.LogicalAnd(x, y);
// Результат: false
```

ПРИМЕР

Логическое И двух вариантов с несовместимыми значениями:

```
x: variant = true;
y: variant = 1;
sum: variant = Variant.LogicalAnd(x, y);
// Результат: VT_EMPTY
```

7.4.15. LogicalOr

Возвращает логическое ИЛИ двух вариантов.

```
variant Variant.LogicalOr(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый вариант. Должен иметь значение типа bool.
y	variant	Второй вариант. Должен иметь значение типа bool.

Возвращаемое значение

Тип - variant.

Логическое ИЛИ параметров x и y. Если любой из параметров содержит значение не типа bool, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

Логическое ИЛИ двух вариантов с логическими значениями:

```
x: variant = true;
y: variant = false;
res: variant = Variant.LogicalOr(x, y);
// Результат: true
```

ПРИМЕР

Логическое ИЛИ двух вариантов с несовместимыми значениями:

```
x: variant = true;
y: variant = 1;
res: variant = Variant.LogicalOr(x, y);
// Результат: VT_EMPTY
```

7.4.16. LT

Сравнивает два варианта аналогично оператору "меньше".

```
variant Variant.LT(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый вариант.
y	variant	Второй вариант.

Возвращаемое значение

Тип - variant.

Результат сравнения x < y:

- true - если операция истинна для значений.
- false - если операция ложна для значений.
- VT_EMPTY - если значения несравнимы или один из операндов - пустое значение (VT_EMPTY).

ПРИМЕР

```
x: variant = 40;
y: variant = 40.0;
res: variant = Variant.LT(x, y);
// Результат: false
```

ПРИМЕР

```
x: variant = 40;
y: variant = "40";
res: variant = Variant.LT(x, y);
// Результат: VT_EMPTY
```

7.4.17. MayConvertTo<T>

Группа функций, которые проверяют, можно ли конвертировать значение типа variant в элементарный тип.

```
uint1 Variant.MayConvertToInt1(variant x)
uint1 Variant.MayConvertToInt2(variant x)
uint1 Variant.MayConvertToInt4(variant x)
uint1 Variant.MayConvertToInt8(variant x)
```

```
uint1 Variant.MayConvertToUint1(variant x)
uint1 Variant.MayConvertToUint2(variant x)
uint1 Variant.MayConvertToUint4(variant x)
uint1 Variant.MayConvertToUint8(variant x)
uint1 Variant.MayConvertToFloat(variant x)
uint1 Variant.MayConvertToDouble(variant x)
uint1 Variant.MayConvertToBool(variant x)
uint1 Variant.MayConvertToString(variant x)
```

Параметры

Параметр	Тип	Описание
x	variant	Значение, для которого будем проверять возможность конвертации.

Возвращаемое значение

Тип - uint1.

Одно из значений:

- 0x00 (MAY_CONVERTED) - конвертация возможна.
- 0x80 (CAN_NOT_CONVERTED) - конвертация невозможна.
- 0x01 (MAY_CONVERTED_WITH_LOSS) - возможна конвертация с потерей точности.
- 0x02 (MAY_CONVERTED_WITH_CUT) - возможна конвертация с обрезкой.

Если одновременно возможна конвертация с обрезкой и потерей точности, то вернется 0x02 (MAY_CONVERTED_WITH_CUT) - возможна конвертация с обрезкой.

ПРИМЕР

Проверяем, можно ли конвертировать значение в int1:

```
V1: variant = 128;
ResultConvert: uint1 = Variant.MayConvertToInt1(V1);
// Результат: 0x02 – возможна конвертация с обрезкой (до 127)
```

ПРИМЕР

Проверяем, можно ли конвертировать значение в int1

```
V2: variant = 999.089f;
ResultConvert: uint1 = Variant.MayConvertToInt1(V2);
//Результат: 0x01 - возможна конвертация с потерей точности (до 100)
```

ПРИМЕР

Проверяем, можно ли конвертировать значение в string:

```
V3: variant = "Моя строка";
ResultConvert: uint1 = Variant.MayConvertToString(V3);
// Результат: 0x00 - конвертация возможна
```

ПРИМЕР

Проверяем, можно ли конвертировать значение в uint1:

```
V3: variant = "Моя строка";
ResultConvert: uint1 = Variant.MayConvertToUint1(V3);
// Результат: 0x80 – конвертация невозможна (строку нельзя конвертировать в число)
```

ПРИМЕР

Проверяем, можно ли конвертировать значение в bool:

```
V4: variant = true;
ResultConvert: uint1 = Variant.MayConvertToBool(V4);
// Результат: 0x00 – конвертация возможна
```

7.4.18. Mul

Возвращает произведение двух варинатов.

```
variant Variant.Mul(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый множитель.
y	variant	Второй множитель.

Возвращаемое значение

Тип - variant.

Произведение x и y. Если типы значений несовместимы, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

Произведение двух вариантов с целыми значениями:

```
x: variant = 40;
y: variant = 10;
res: variant = Variant.Mul(x, y);
// Результат: 400
```

ПРИМЕР

Произведение двух вариантов с несовместимыми значениями:

```
x: variant = 40;
y: variant = "10";
res: variant = Variant.Mul(x, y);
// Результат: VT_EMPTY
```

7.4.19. NE

Сравнивает два варианта аналогично оператору "не равно".

```
variant Variant.NE(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый вариант.
y	variant	Второй вариант.

Возвращаемое значение

Тип - variant.

Результат сравнения x != y:

- true - если операция истинна для значений.
- false - если операция ложна для значений.
- VT_EMPTY - если значения несравнимы или один из операндов - пустое значение (VT_EMPTY).

ПРИМЕР

```
x: variant = 40;
y: variant = 40.0;
res: variant = Variant.NE(x, y);
// Результат: false
```

ПРИМЕР

```
x: variant = 40;
y: variant = "40";
res: variant = Variant.NE(x, y);
// Результат: VT_EMPTY
```

7.4.20. NEG

Возвращает значение указанного варианта с противоположным знаком.

```
variant Variant.NEG(variant x)
```

Параметры

Параметр	Тип	Описание
x	variant	Вариант. Должен иметь числовое значение.

Возвращаемое значение

Тип - variant.

Значение x с противоположным знаком. Если x имеет не числовой тип, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

```
x: variant = 0.1;
res: variant = Variant.NEG(x);
// Результат: -0.1
```

ПРИМЕР

```
x: variant = true;
res: variant = Variant.NEG(x);
// Результат: VT_EMPTY
```

7.4.21. NOT

Возвращает логическое отрицание указанного варианта.

```
variant Variant.NOT(variant x)
```

Параметры

Параметр	Тип	Описание
x	variant	Вариант. Должен иметь значение типа bool.

Возвращаемое значение

Тип - variant.

Логическое отрицание варианта x. Если x имеет тип значения не bool, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

Логическое отрицание варианта с логическим значением:

```
x: variant = true;
res: variant = Variant.NOT(x);
// Результат: false
```

ПРИМЕР

Логическое отрицание варианта не с логическим значением:

```
x: variant = 1;
res: variant = Variant.NOT(x);
// Результат: VT_EMPTY
```

7.4.22. Or

Возвращает побитовое ИЛИ двух вариантов.

```
variant Variant.Or(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый вариант. Должен иметь целочисленное значение.

Параметр	Тип	Описание
y	variant	Второй вариант. Должен иметь целочисленное значение.

Возвращаемое значение

Тип - variant.

Побитовое ИЛИ параметров x и y. Если любой из параметров содержит не целочисленное значение, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

Побитовое ИЛИ двух вариантов с целочисленными значениями:

```
x: variant = 12; // В двоичной записи - 1100
y: variant = 21; // В двоичной записи - 10101
res: variant = Variant.Or(x, y);
// Результат: 29 (в двоичной записи - 11101)
```

ПРИМЕР

Побитовое ИЛИ двух вариантов с несовместимыми значениями:

```
x: variant = 12;
y: variant = 1.0;
res: variant = Variant.Or(x, y);
// Результат: VT_EMPTY
```

7.4.23. Sub

Возвращает разность двух вариантов.

```
variant Variant.Sub(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Уменьшаемое.
y	variant	Вычитаемое.

Возвращаемое значение

Тип - variant.

Разность x - y. Если типы значений несовместимы, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

Разность двух вариантов с целыми значениями:

```
x: variant = 40;
y: variant = 10;
res: variant = Variant.Sub(x, y);
// Результат: 30
```

ПРИМЕР

Разность двух вариантов с несовместимыми значениями:

```
x: variant = 40;
y: variant = "10";
res: variant = Variant.Sub(x, y);
// Результат: VT_EMPTY
```

7.4.24. To<T>

Группа функций, которые для значения типа variant возвращают её значение, конвертированное в элементарный тип.

```
int1 Variant.ToInt1(variant x, int1 def_value)
int2 Variant.ToInt2(variant x, int2 def_value)
int4 Variant.ToInt4(variant x, int4 def_value)
int8 Variant.ToInt8(variant x, int8 def_value)
uint1 Variant.ToUint1(variant x, uint1 def_value)
uint2 Variant.ToUint2(variant x, uint2 def_value)
uint4 Variant.ToUint4(variant x, uint4 def_value)
uint8 Variant.ToUint8(variant x, uint8 def_value)
float Variant.ToFloat(variant x, float def_value)
double Variant.ToDouble(variant x, double def_value)
bool Variant.ToBool(variant x, bool def_value)
string Variant.ToString(variant x, string def_value)
```

Параметры

Параметр	Тип	Описание
x	variant	Вариант, который нужно конвертировать.
def_value	все кроме variant	Значение, которое вернётся, если конвертировать не получится.

Возвращаемое значение

Тип - зависит от функции.

Значение варианта x, приведённое к элементарному типу; def_value - если конвертировать не получится.

ПРИМЕР

Конвертировать переменную типа variant в переменную элементарного типа uint1:

```
V1: variant = 120;
Result1: uint1 = Variant.ToUint1(V1, 0);
```

ПРИМЕР

Конвертировать переменную типа variant в переменную типа float:

```
V2: variant = 999.089f;
Result1: float = Variant.ToFloat(V2, 0.0f);
```

ПРИМЕР

Конвертировать переменную типа variant в переменную типа string:

```
V3: variant = "Моя строка";
Result2: string = Variant.ToString(V3, "");
```

ПРИМЕР

Конвертировать переменную типа variant в переменную типа bool:

```
V4: variant = true;
Result3: bool = Variant.ToBool(V4, false);
```

7.4.25. Xor

Возвращает побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ двух вариантов.

```
variant Variant.Xor(variant x, variant y)
```

Параметры

Параметр	Тип	Описание
x	variant	Первый вариант. Должен иметь целочисленное значение.
y	variant	Второй вариант. Должен иметь целочисленное значение.

Возвращаемое значение

Тип - variant.

Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ параметров x и y. Если любой из параметров содержит не целочисленное значение, вернётся пустое значение (VT_EMPTY).

ПРИМЕР

Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ двух вариантов с целочисленными значениями:

```
x: variant = 12; // В двоичной записи - 1100
y: variant = 21; // В двоичной записи - 10101
res: variant = Variant.Xor(x, y);
// Результат: 25 (в двоичной записи - 11001)
```

ПРИМЕР

Побитовое ИЛИ двух вариантов с несовместимыми значениями:

```
x: variant = 12;
y: variant = 1.0;
res: variant = Variant.Xor(x, y);
// Результат: VT_EMPTY
```

8. Приложения

Приложение А: Формат даты/времени

В некоторых функциях обработки времени можно указать формат, согласно которому будет формироваться или разбираться строка, описывающая дату/время. Формат представляет собой строку, в которой с помощью флагов форматирования описано расположение компонентов даты/времени в строке.

Флаги форматирования

Флаг	Описание	Пример
%a	Сокращённое название дня недели.	**«пн»** → понедельник
%A	Название дня недели.	**«понедельник»** → понедельник
%b	Сокращённое название месяца.	**«янв»** → январь
%B	Название месяца.	**«январь»** → январь
%c	Принятое представление даты/времени в текущей локали.	
%C	Первые две цифры в записи года.	2015 → **«20»**
%d	Дата месяца от 01 до 31.	**«01»** → 1 число **«15»** → 15 число
%D	Аналогичен %m%d%y.	
%e	Аналогичен %d, но ведущий ноль заменяется на пробел.	**« 1»** → 1 число **«20»** → 20 число
%f	Дробные доли секунды, отображаются даже если равны нулю.	**«12:30:15.000000»**
%F	Дробные доли секунды, отображаются только если не равны нулю.	**«12:30:15»** **«01:02:03.012345»**
%h	Аналогичен %b.	
%H	Час от 00 до 23.	
%I	Час от 01 до 12.	**«20»** → 8
%j	Номер дня в году от 001 до 365 (всегда предполагается, что год високосный).	**«060»** → 29 февраля

Флаг	Описание	Пример
%k	Час от 0 до 23.	
%I	Час от 1 до 12.	
%m	Номер месяца от 01 до 31.	**«01»** → январь
%M	Минуты от 00 до 59.	
%s	Секунды, включая дробные доли секунд.	**«59.000000»**
%S	Секунды.	**«59»**
%T	Время в 24-часовой нотации. Эквивалентно %H:%M:%S.	**«12:30:00»**
%u	Номер дня недели от 1 до 7, где 1 - понедельник.	**«2»** → вторник
%U	Номер недели в году от 00 до 53. Первое воскресенье в году считается первым днём недели 01. Если год начинается не с воскресенья, дни до первого воскресенья считаются неделей 00.	
%V	Номер недели в формате ISO 8601:1988, значение в диапазоне от 01 до 53. Неделей 01 считается первая неделя января, в которой минимум 4 дня относятся к этому году; если меньше - первой считается следующая за ней неделя.	
%w	Номер дня недели от 0 до 6, где 0 - воскресенье.	**«1»** → понедельник
%W	Номер недели в году от 00 до 53, где понедельник - это первый день недели 01.	
%x	Формат даты, принятый для текущей локали.	en_us → **«10/31/2005»** ru_ru → **«31.10.2005»**
%y	Две цифры года.	**«01»** → 2001
%Y	Четыре цифры года.	**«2001»** → 2001

Примеры

- %Y-%B-%d → **«2005-апрель-01»**
- %Y%m%d → **«20050401»**
- %Y-%m-%d %H:%M:%S%F → **«2005-01-02 12:30:00»**

История изменений

1.1

В этой версии - только внутренние изменения. Функциональных изменений нет.

1.1.1

В этой версии - только внутренние изменения. Функциональных изменений нет.

1.1.2

В этой версии - только внутренние изменения. Функциональных изменений нет.

1.1.3

В этой версии - только внутренние изменения. Функциональных изменений нет.

1.1.4

В этой версии - только внутренние изменения. Функциональных изменений нет.

Изменения документации

Редакция 2

Добавлены описания патчей 1-4. Содержимое документа не изменилось.

1.0

1.0.1

Внутренние изменения, функциональных изменений нет.

1.0.2

Новые возможности:

- Добавлена функция String.IsValidFormat, которая позволяет проверить, является ли указанная строка корректной строкой формата printf (стр. 76).

  С её помощью можно проверять корректность строк, передаваемых в качестве описания формата для функции String.ToString.

- Добавлена функция DateTime.IsLeapYear, которая позволяет определить, является ли указанный год високосным (стр. 103).

- Добавлена функция DateTime.DaysInMonth, которая позволяет получить количество дней в указанном месяце указанного года (стр. 102).

Исправленные ошибки

- Функция DateTime.AddMonths возвращала нулевую метку времени (01.01.1601), если при смещении получалась недопустимая дата: 31 января → 31 февраля.

  Теперь, если в результате смещения получается недопустимая дата, возвращается последний день полученного месяца (с учётом високосного года).

- Если в функцию String.ToString была передана некорректная строка описания формата, вызывалось исключение и скрипт прекращал выполняться.

  Теперь в этом случает исключение не вызывается, функция возвращает пустую строку.

- При совпадении имён деклараций в коде скрипта (например, <declaration_name> : string = "42";) с именами из внешнего окружения (имена верхнего уровня, имена внешних ссылок и пр.) происходила ошибка компиляции.

  Теперь ошибки компиляции не происходит, вместо этого в журнал выводится предупреждение.

1.0.3

Новые возможности:

- Для функций String.IndexOf и String.LastIndexOf добавлены перегрузки, позволяющие указать позицию начала поиска подстроки.
- Для функции String.ToLocalizedString добавлены перегрузки с возможностью указания формата.

Улучшение:

- В экспоненциальной записи вещественного числа можно не указывать знак: запись без знака 12.3e4 эквивалентна записи со знаком плюс 12.3e+4.

1.0.4

Изменение:

- Функции String.ToString и String.ToLocalizedString возвращают строку в кодировке UTF-8 вместо ASCII.

Изменения документации

Редакция 2

Внутренние изменения, содержимое документа не изменилось.

Редакция 3

Внесены изменения, произошедшие в патчах 2-4.